High Performance Software Defined Radio (HPSDR)

John Melton

G0ORX/N6LYT

john.d.melton@googlemail.com

HPSDR

What's Is HPSDR All About?

A group of worldwide enthusiasts interested in developing an open source hardware and software project intended as a "next generation" Software Defined Radio for use by Radio Amateurs and Short Wave Listeners.

HPSDR

Who are TAPR?

Tucson Amateur Packet Radio

TAPR is a community that provides leadership and resources to radio amateurs for the purpose of advancing the radio art.

For HPSDR they typically help with the development of the hardware by funding the cost of building Alpha and Beta boards. They will then make a production run of some number of boards that they will sell through their web site to recoup their costs for the development.

HPSDR

TAPR Open Hardware License

The TAPR Open Hardware License ("OHL") provides a framework for hardware projects that is similar to the one used for Open Source software.

This isn't as straight-forward as it seems because legal concepts that work well for software (such as copyright and copyleft) don't neatly fit when dealing with hardware products and the documentation used to create them.

Noncommercial OHL restricts use of design for commercial use.

HPSDR

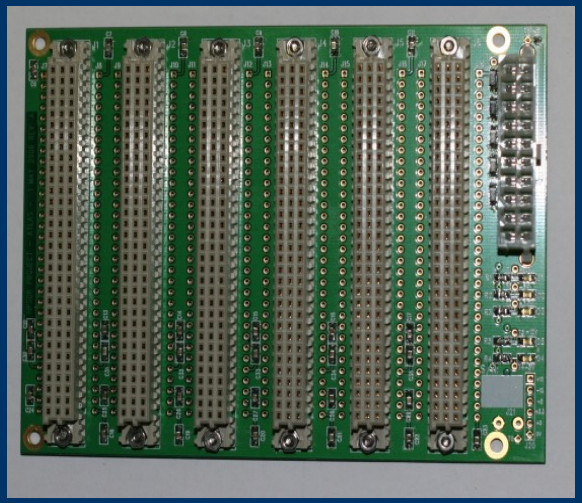
Software Defined Radio (SDR)

An SDR is a communications device where the typical hardware components such as mixers, filters, amplifiers, modulators/demodulators, detectors, etc. are implemented in software.

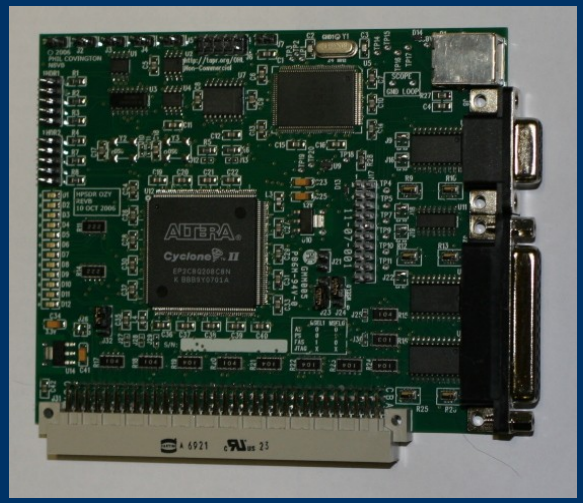
Such a design produces a radio which can receive and transmit widely different radio protocols (sometimes referred to as waveforms) based solely on the software used.

HPSDR

Basic Hardware Components



Atlas
(Backplane)



Ozy/Magister/Metis
(I/O)

Mercury (Rx)

Penelope (Tx)

HPSDR

Firmware Development

Written in VHDL

Quartus II Web Edition FPGA design software

- Free download from Altera (www.altera.com)
  - Windows and Linux versions.

HPSDR

CIC Filter

```
module cic( clock, in_strobe, out_strobe, in_data, out_data );

//design parameters
parameter STAGES = 3;
parameter DECIMATION = 16;
parameter IN_WIDTH = 18;

//computed parameters
//ACC_WIDTH = IN_WIDTH + Ceil(STAGES * Log2(DECIMATION))
//OUT_WIDTH = IN_WIDTH + Ceil(Log2(DECIMATION) / 2)
parameter ACC_WIDTH = IN_WIDTH  + 12;
parameter OUT_WIDTH = IN_WIDTH  + 2;

input clock;
input in_strobe;
output reg out_strobe;
input signed [IN_WIDTH-1:0] in_data;
output signed [OUT_WIDTH-1:0] out_data;

//-------------------------------------------------------------------------
//                          control
//-------------------------------------------------------------------------
reg [15:0] sample_no;
initial sample_no = 15'd0;


always @(posedge clock)
  if (in_strobe)
    begin
    if (sample_no == (DECIMATION-1))
      begin
      sample_no <= 0;
      out_strobe <= 1;
      end
    else
      begin
      sample_no <= sample_no + 8'd1;
      out_strobe <= 0;
      end
    end

  else
    out_strobe <= 0;


//-------------------------------------------------------------------------
//                          stages
//-------------------------------------------------------------------------
wire signed [ACC_WIDTH-1:0] integrator_data [0:STAGES];
wire signed [ACC_WIDTH-1:0] comb_data [0:STAGES];


assign integrator_data[0] = in_data;
assign comb_data[0] = integrator_data[STAGES];


genvar i;
generate
  for (i=0; i<STAGES; i=i+1)
    begin : cic_stages

    cic_integrator #(ACC_WIDTH) cic_integrator_inst(
      .clock(clock),
     .strobe(in_strobe),
      .in_data(integrator_data[i]),
      .out_data(integrator_data[i+1])
    );


    cic_comb #(ACC_WIDTH) cic_comb_inst(
      .clock(clock),
      .strobe(out_strobe),
      .in_data(comb_data[i]),
      .out_data(comb_data[i+1])
      );
    end
endgenerate



//-------------------------------------------------------------------------
//                          output rounding
//-------------------------------------------------------------------------
assign out_data = comb_data[STAGES][ACC_WIDTH-1:ACC_WIDTH-OUT_WIDTH] +
  {{(OUT_WIDTH-1){1'b0}}, comb_data[STAGES][ACC_WIDTH-OUT_WIDTH-1]};



endmodule
```

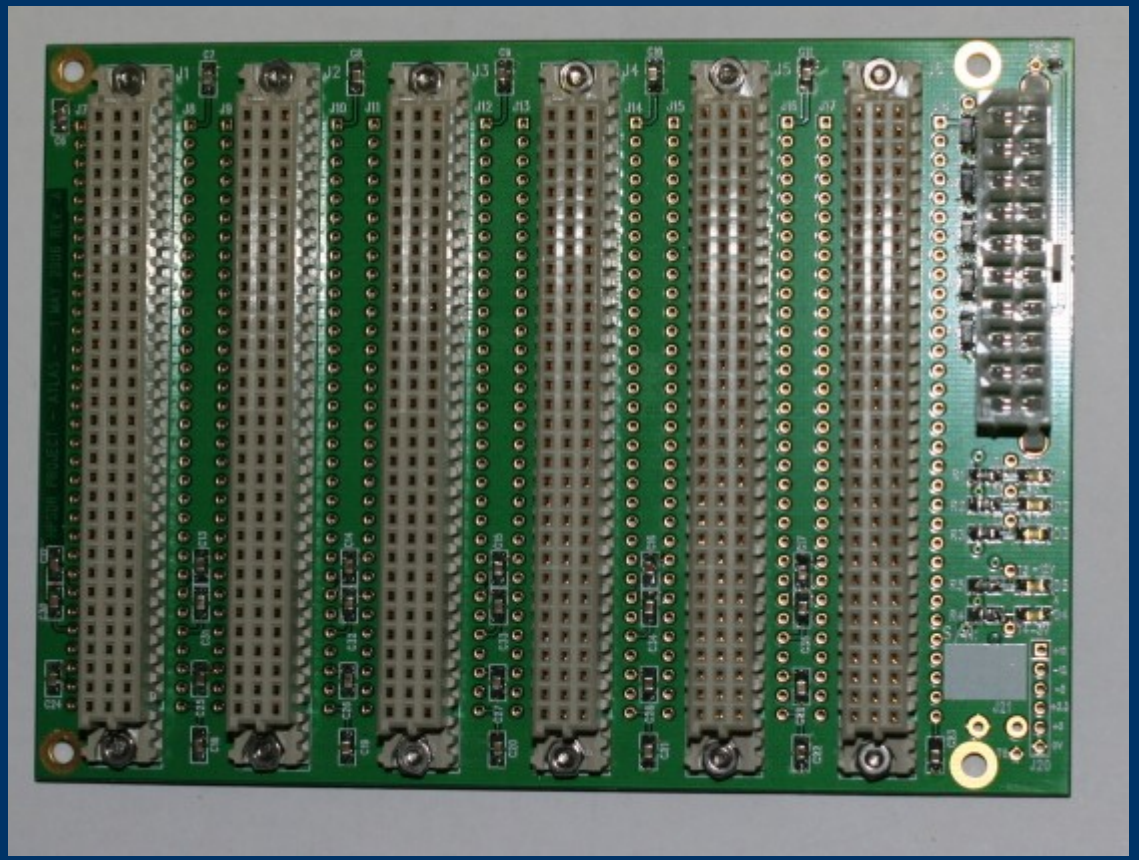
HPSDR

Atlas

Passive backplane.

6 DIN connectors.

20 Pin ATX Power connector.



HPSDR

OZY/Magister

Interface controller.

Cypress FX2 USB 2.0 controller

Altera Cyclone II FPGA.

Interface for PTT and CW paddles.

HPSDR

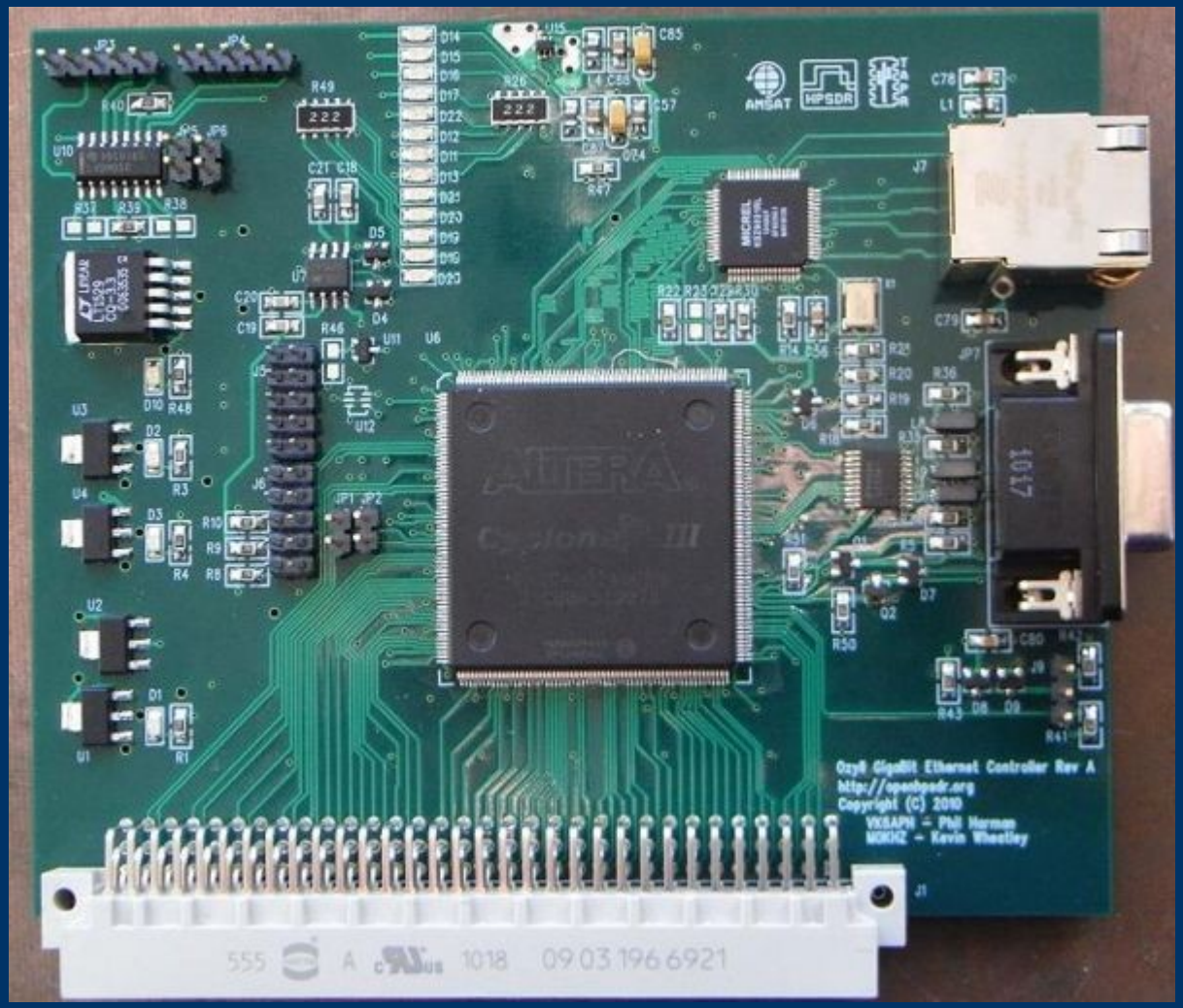
Metis (OzyII)

Interface controller.

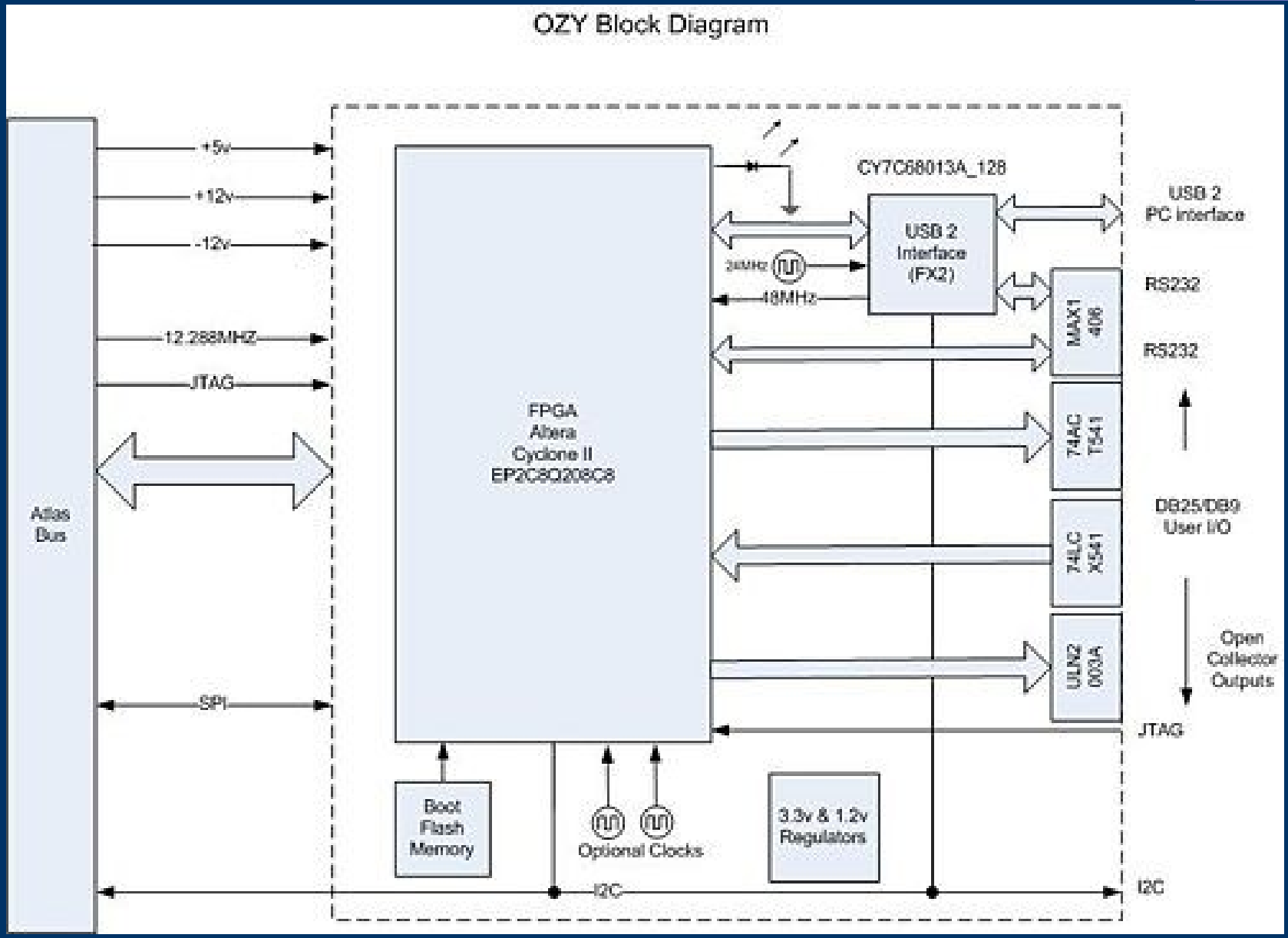
10/100/1000 ethernet.

Altera Cyclone III FPGA.

Interface for PTT and CW paddles.



HPSDR


OZY Block Diagram
+5v
+12v
-12v
12.288MHz
JTAG
Atlas Bus
SPI
FPGA Altera Cyclone II EP2C8Q208C8
CY7C68013A_128
USB 2 Interface (FX2)
24MHz
48MHz
USB 2 PC Interface
RS232
MAX1 406
RS232
74AC T541
DB25/DB9 User I/O
74LC X541
ULN2 003A
Open Collector Outputs
JTAG
Boot Flash Memory
Optional Clocks
3.3v & 1.2v Regulators
I2C
I2C

HPSDR

Both OZY and Magister can also be configured to look like a USB-Blaster for loading FPGA code to Mercury and Penelope.

HPSDR

Mercury

0-65 MHz direct sampling receiver.

Linear Technology LTC2208 130MSPS 16-bit A/D converter.

Altera Cyclone III FPGA.

Digital Down Conversion to 48K, 96K or 192K.

HPSDR

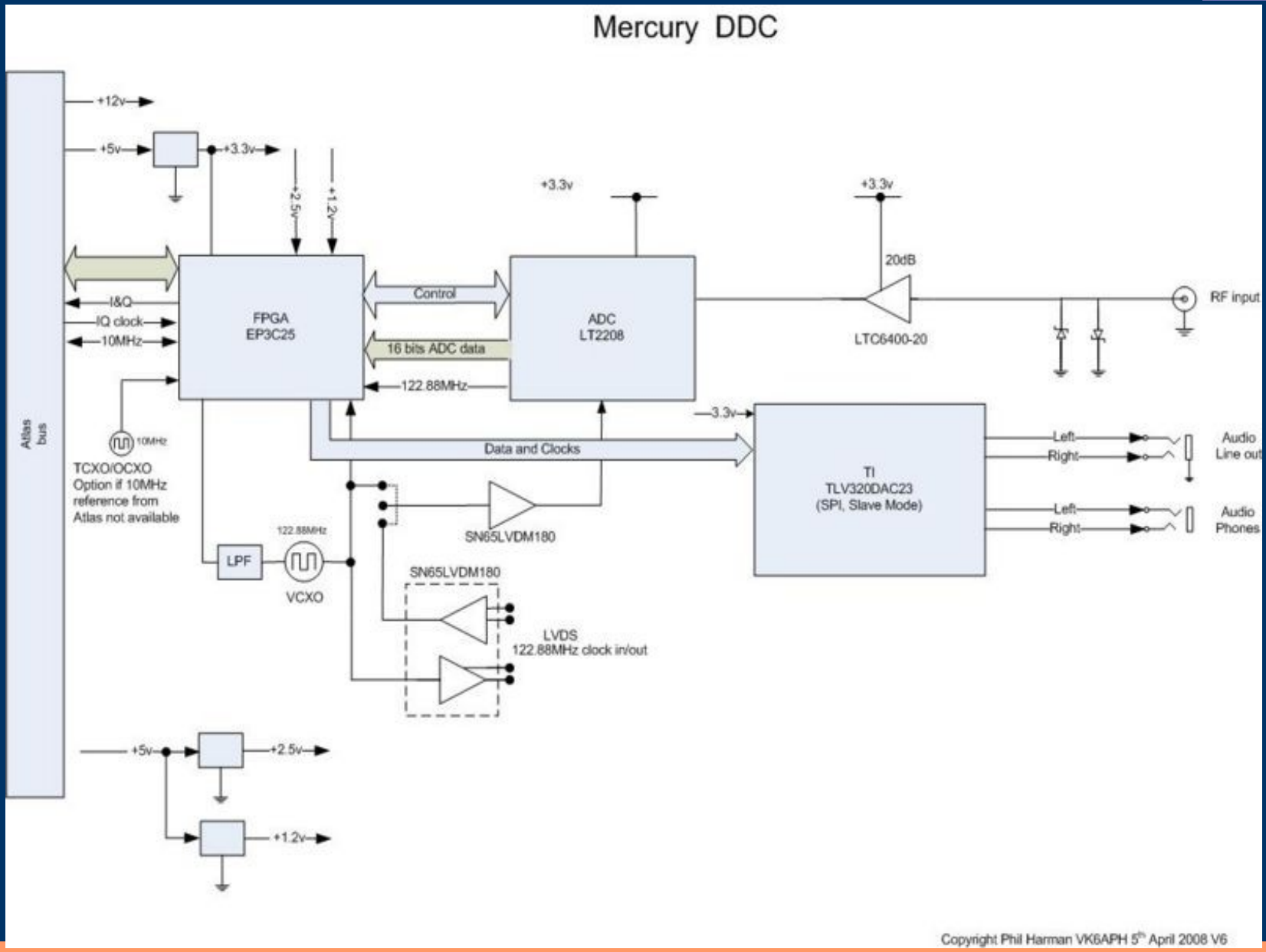
Mercury Performance

ADC overload: -12dBm (preamp on), +8dBm (preamp off)

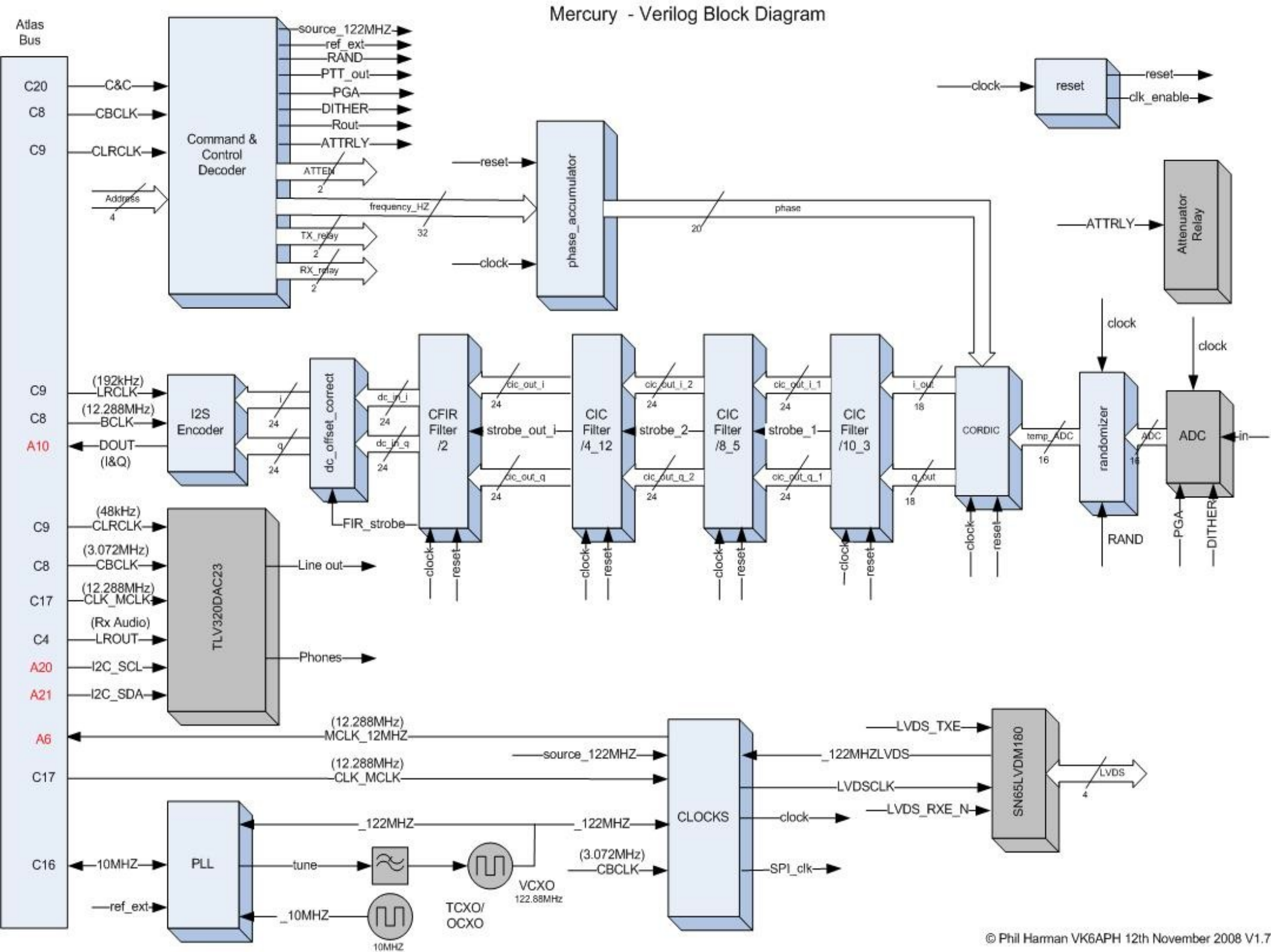
MDS (500Hz), 160m - 6m: -138dBm (preamp on), -118dBm (preamp off)

IP3 equivalent (independent of spacing): +33dBm (preamp on), >+50dBm (preamp off)

HPSDR


Mercury DDC
+12v
+5v
+3.3v
+2.5v
+1.2v
+3.3v
+3.3v
20dB
I&Q
IQ clock
10MHz
FPGA
EP3C25
Control
16 bits ADC data
122.88MHz
ADC
LT2208
LTC6400-20
RF input
Atlas bus
10MHz
TCXO/OCXO
Option if 10MHz
reference from
Atlas not available
Data and Clocks
3.3v
TI
TLV320DAC23
(SPI, Slave Mode)
Left
Right
Audio
Line out
Left
Right
Audio
Phones
122.88MHz
LPF
VCXO
SN65LVDM180
SN65LVDM180
LVDS
122.88MHz clock in/out
+5v
+2.5v
+1.2v
Copyright Phil Harman VK6APH 5th April 2008 V6

Mercury - Verilog Block Diagram



HPSDR

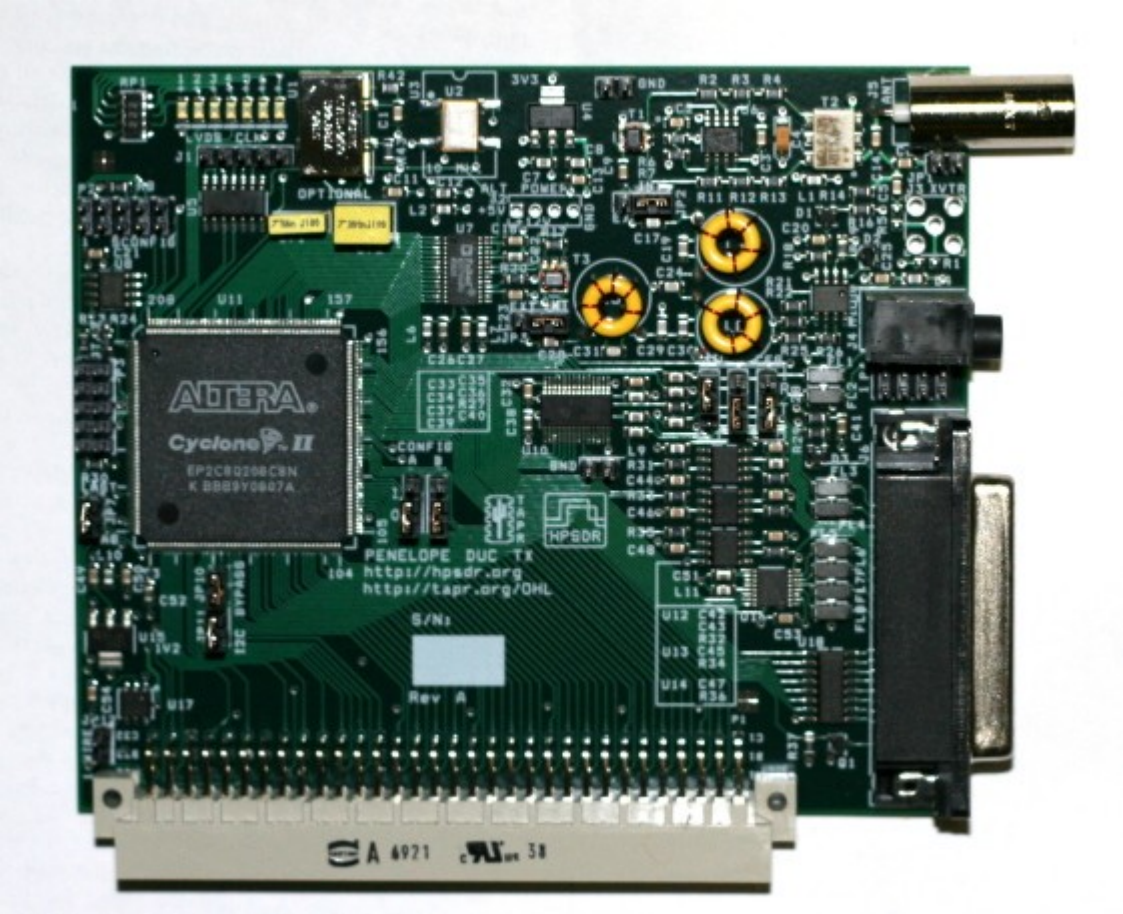
Penelope

½ watt transmitter/exciter board.

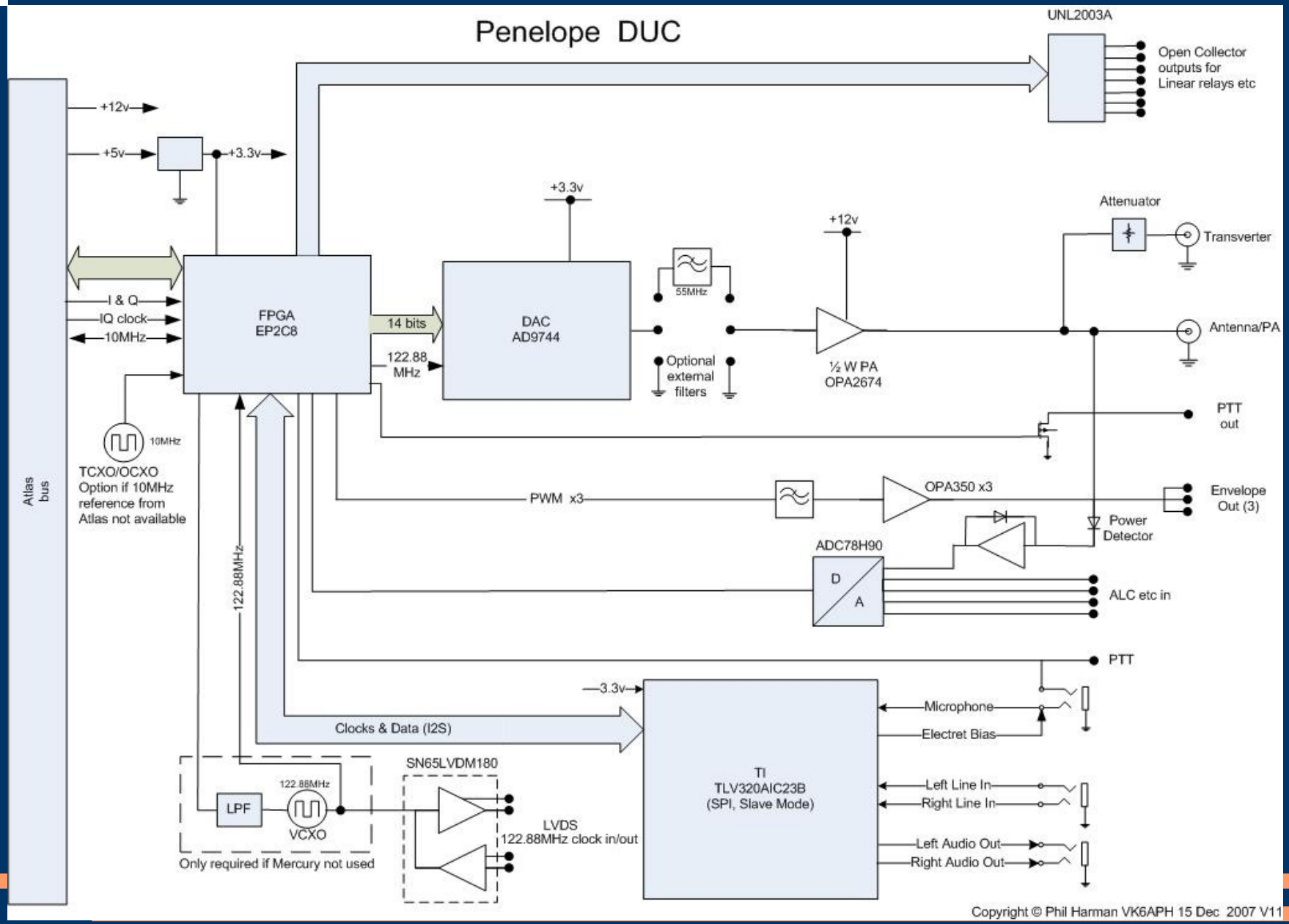
Digital Up Conversion.

Altera Cyclone II FPGA.

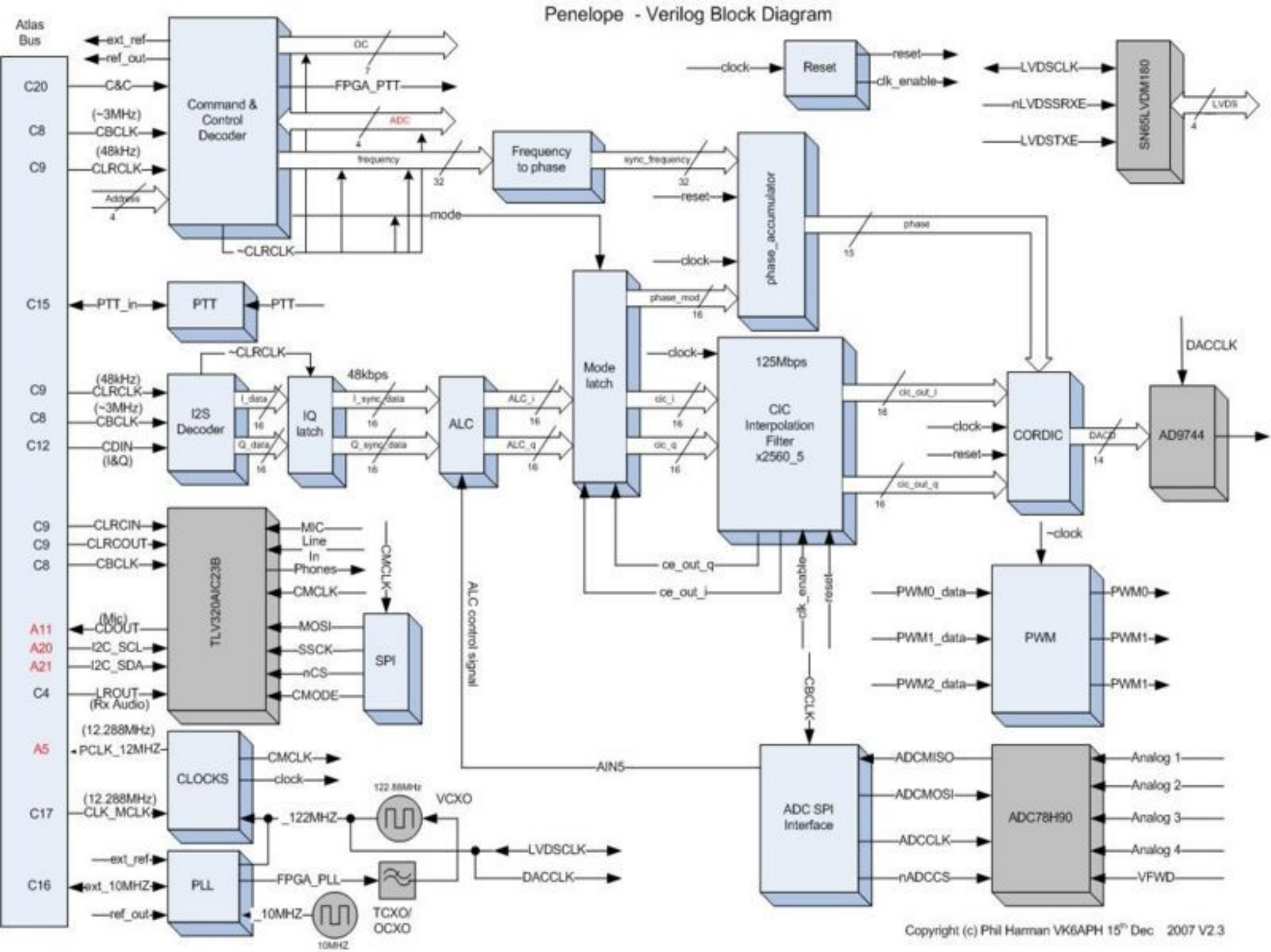
Microphone input and A to D converter.



HPSDR


Penelope DUC
UNL2003A
Open Collector outputs for Linear relays etc
+12v
+5v
+3.3v
+3.3v
Attenuator
Transverter
+12v
I & Q
IQ clock
10MHz
FPGA EP2C8
14 bits
122.88 MHz
DAC AD9744
55MHz
Optional external filters
½ W PA OPA2674
Antenna/PA
PTT out
Atlas bus
10MHz
TCXO/OCXO Option if 10MHz reference from Atlas not available
PWM x3
OPA350 x3
Envelope Out (3)
Power Detector
ADC78H90
D
A
ALC etc in
122.88MHz
PTT
3.3v
Clocks & Data (I2S)
Microphone
Electret Bias
SN65LVDM180
122.88MHz
LPF
VCXO
Only required if Mercury not used
LVDS
122.88MHz clock in/out
TI
TLV320AIC23B
(SPI, Slave Mode)
Left Line In
Right Line In
Left Audio Out
Right Audio Out
Copyright © Phil Harman VK6APH 15 Dec 2007 V11

Penelope - Verilog Block Diagram



HPSDR

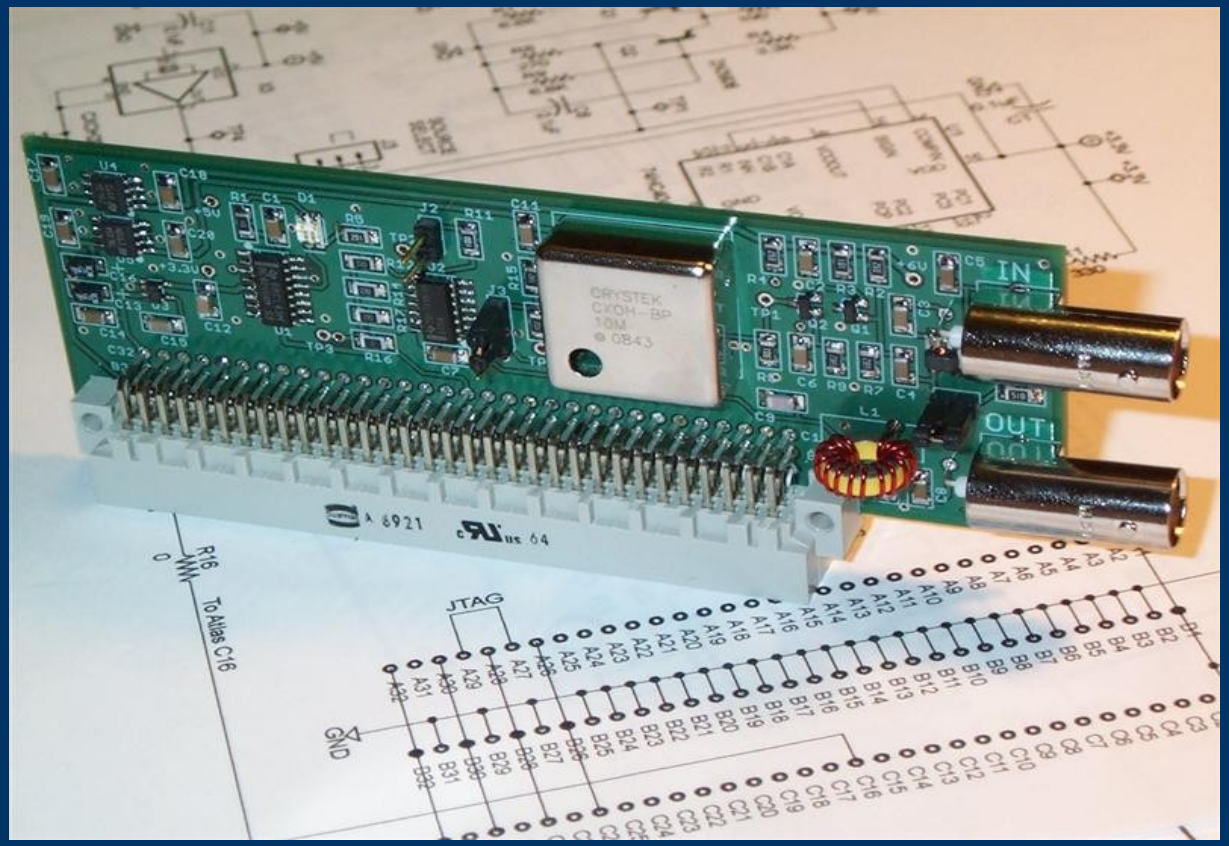
Penelope

WSPR code available to download to the FPGA that will allow Penelope to run in a stand alone.

Requires building code with modified call for the user and frequency to transmit on before downloading to Penelope.

HPSDR

Excalibur



External 10MHz reference clock.

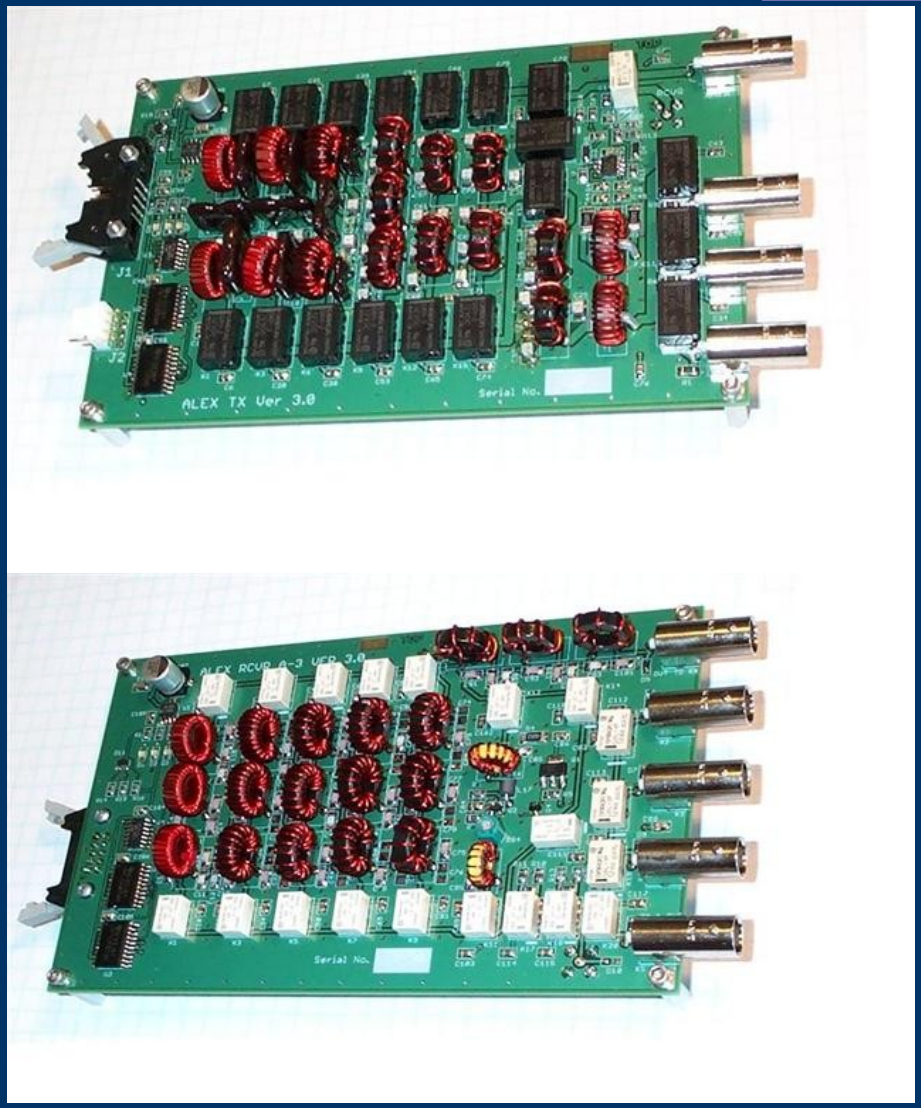
Built in TCXO that is better than the on board oscillators.

Input for external GPS Disciplined Oscillator.

HPSDR

LPU (Linear Power Supply)

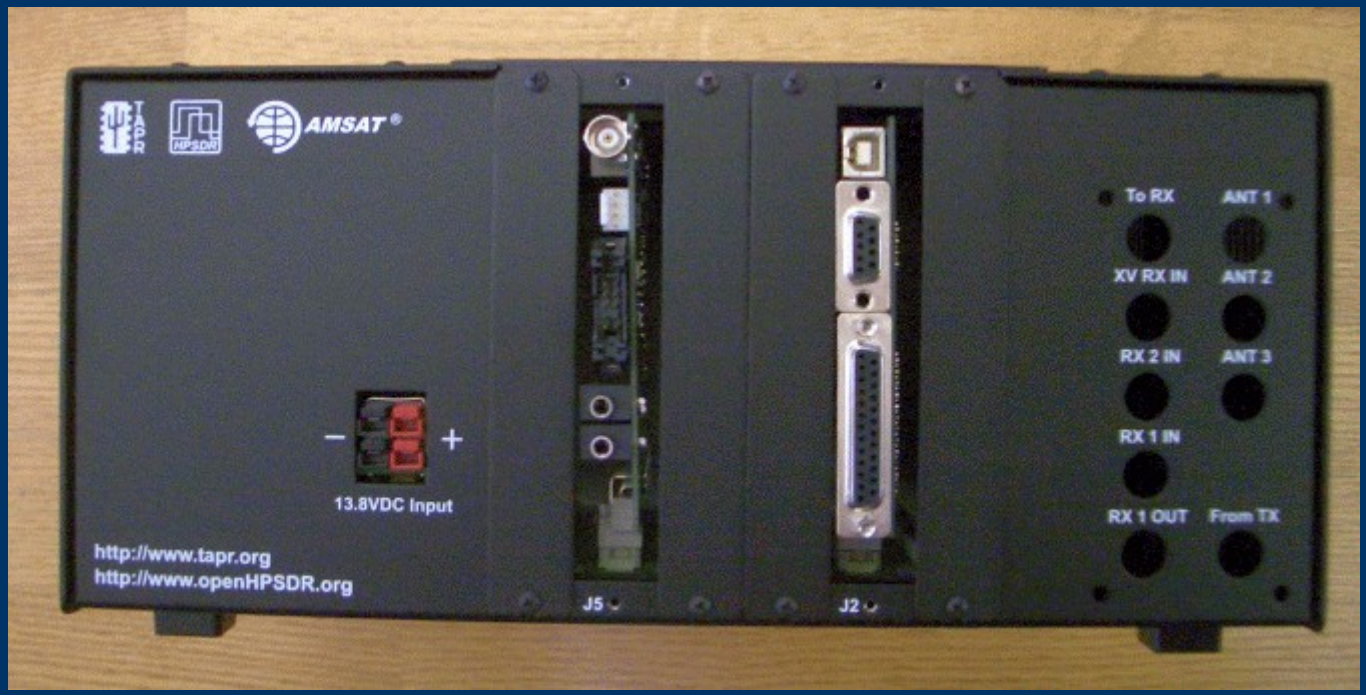
Pennywhistle (16-20 watt PA)



Alex (Filters,Ant switch).

HPSDR

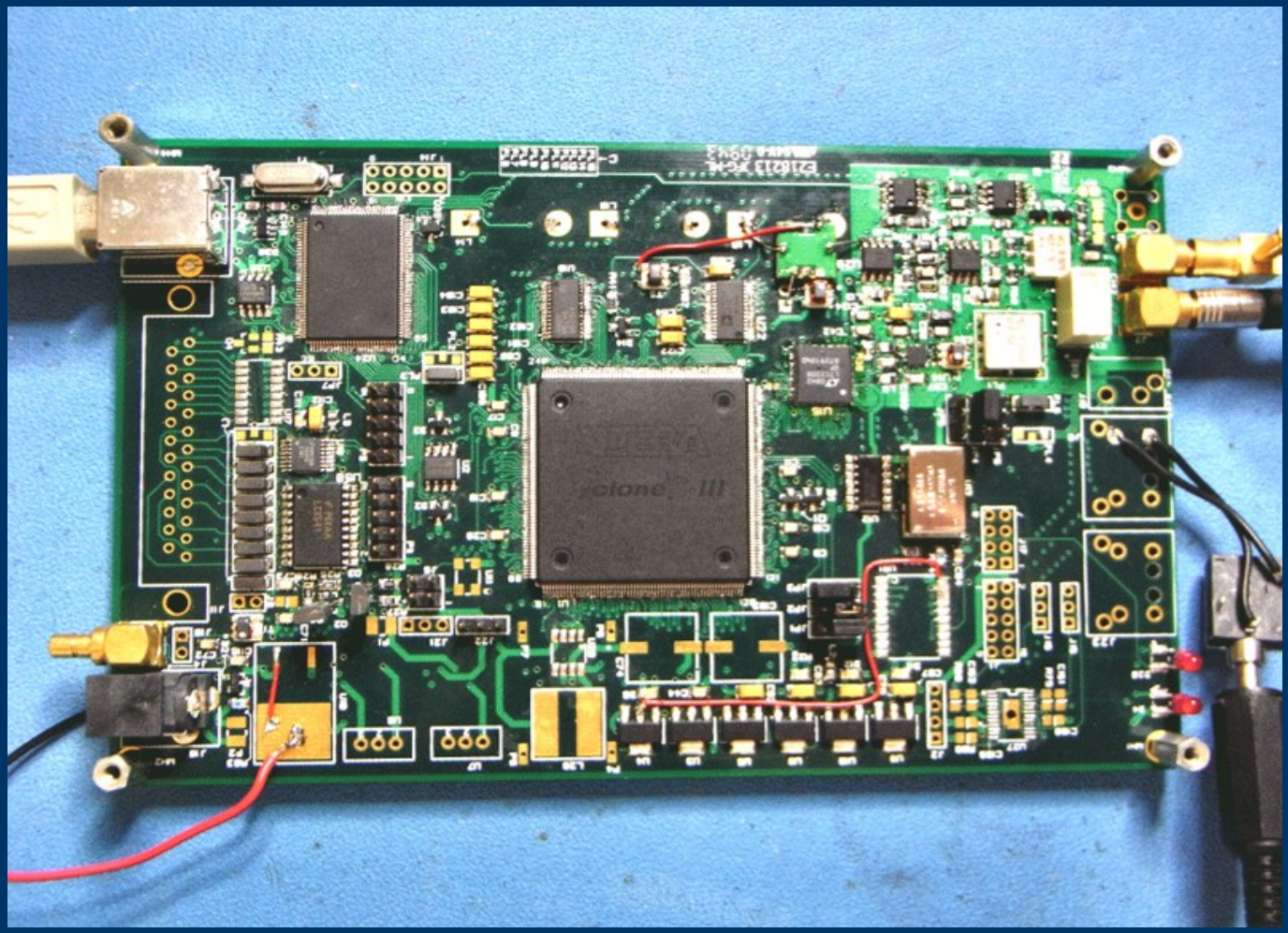
Pandora



HPSDR

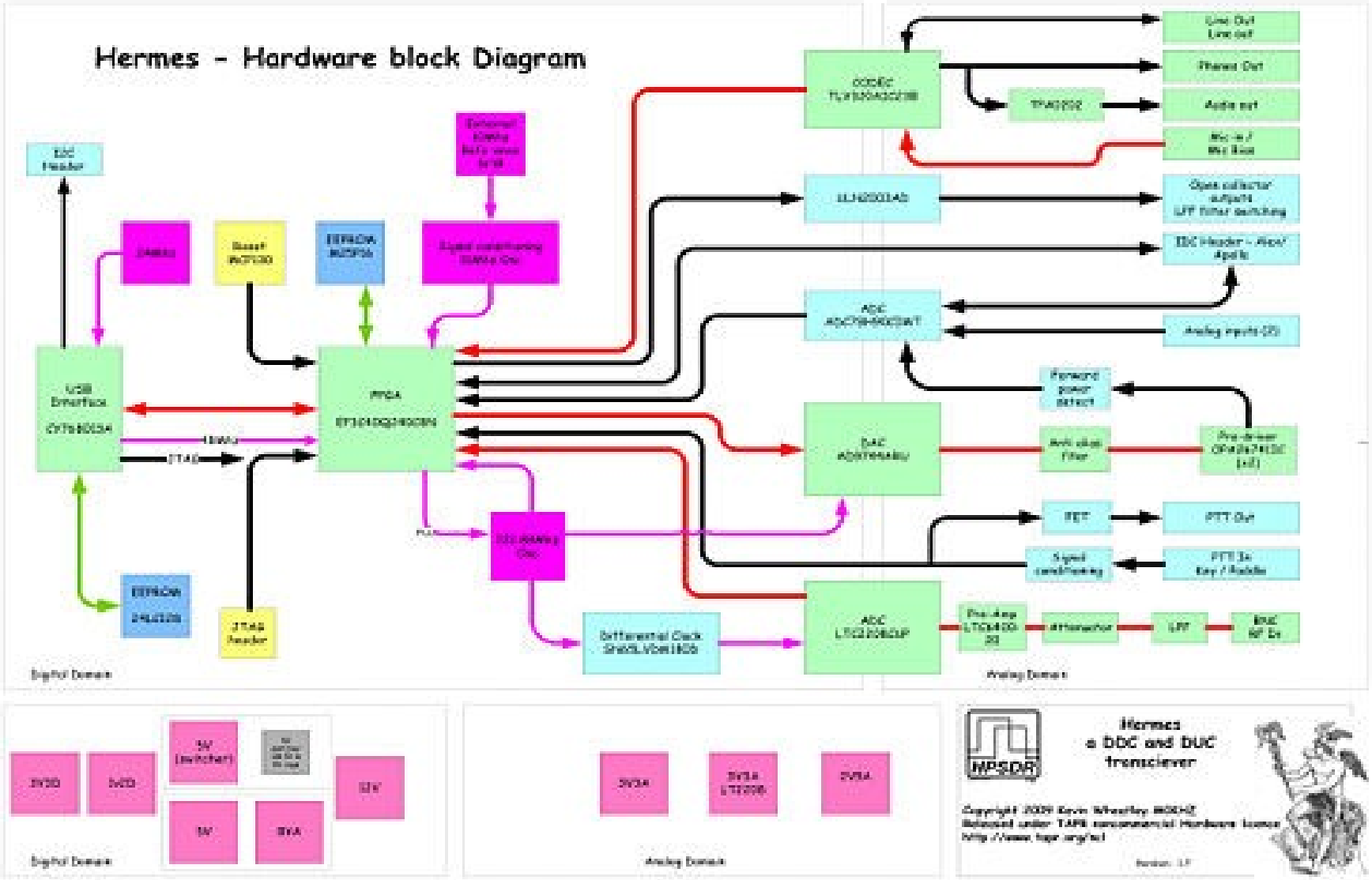
HPSDR

Hermes



- Single board, Metis, Mercury, Penelope.

HPSDR


Hermes - Hardware block Diagram
Digital Domain
Analog Domain
Hermes
a DDC and DUC
transciever

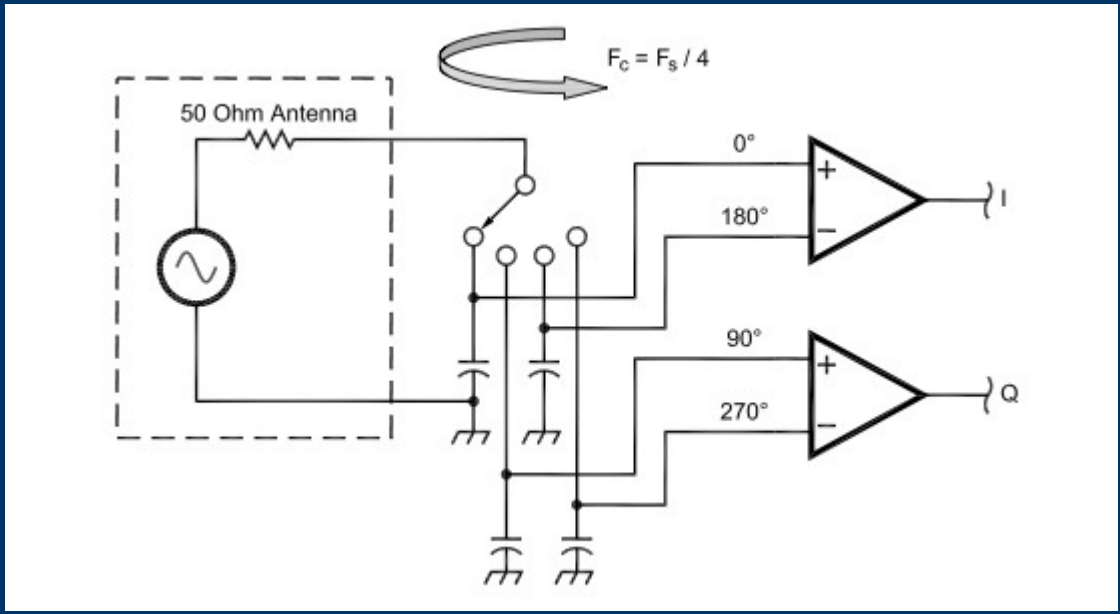
HPSDR

Hermes

- Alpha version with filter board (Apollo)

HPSDR

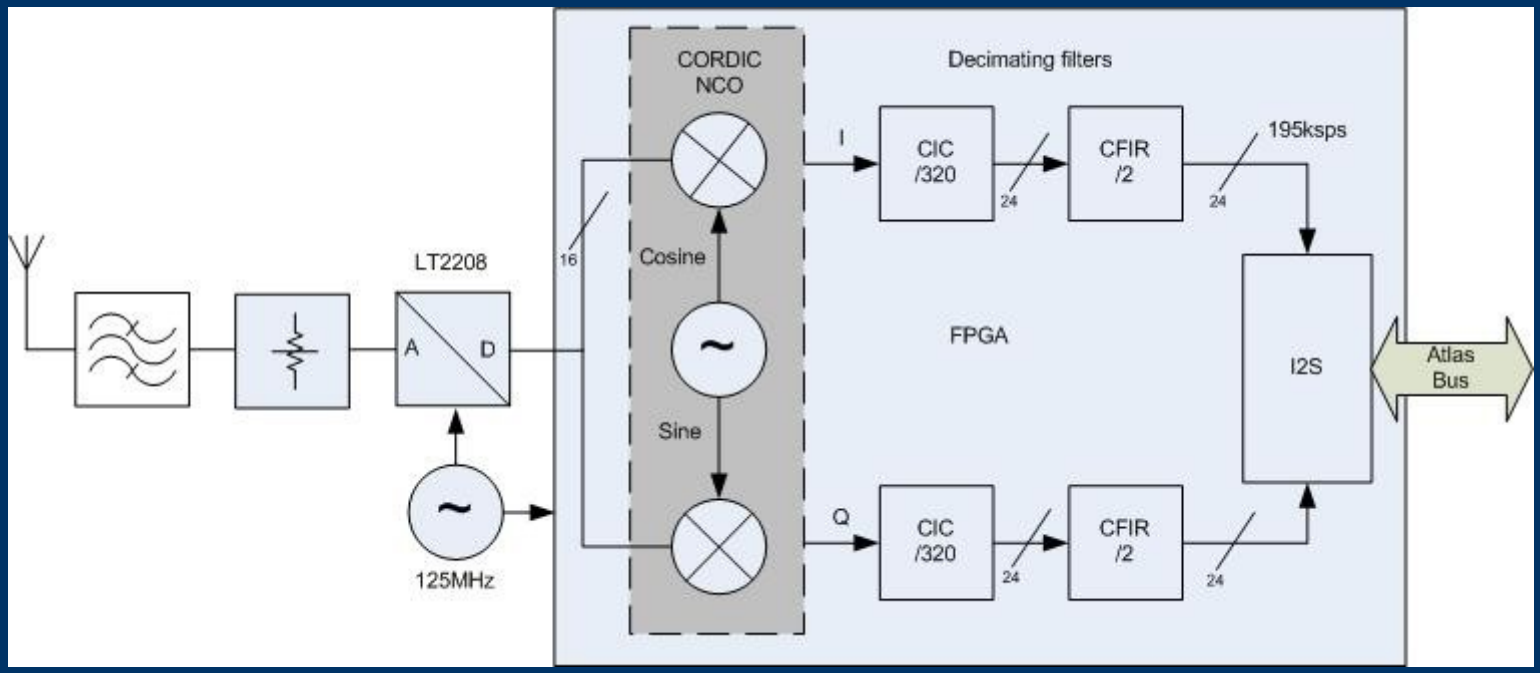
Tayloe Detector (Dan Tayloe)



The switch rotates at the carrier frequency so that each capacitor samples the signal once each revolution. The 0° and 180° capacitors differentially sum to provide the in-phase (I) signal and the 90° and 270° capacitors sum to provide the quadrature (Q) signal.

HPSDR

Mercury Digital Down Conversion



LT2208 ADC sampling at 125MHz

ADC output 0 – 62.5MHz

Decimate by 640

Output = 125MHz/640 = 195ksps (24 bit samples)

HPSDR

Mercury Digital Down Conversion

By decimation we have eased the load on the PC but increased the complexity of the DDC

But there is an additional advantage of decimation!

Every time we decimate by 2 we increase the output SNR by 3dB

HPSDR

I and Q signals

I – in-phase
Q – quadrature (delayed by 90 degrees)

Use FFT to convert complex I/Q time domain signal to frequency domain. Each bin contains spectral density of the signal within that bins bandwidth. If we are sampling at 48000 samples per second and have 4096 bins for the FFT then each bin has a bandwidth of 11.71Hz.

Further filtering and processing allows us to demodulate the signals.

Inverse FFT used to convert the frequency domain signal back to time domain

- A Software-Defined Radio for the Masses, Part 1
  http://www.flex-radio.com/Data/Doc/qex1.pdf

- Quadrature Signals: Complex, But Not Complicated
  http://www.dspguru.com/dsp/tutorials/quadrature-signals

HPSDR

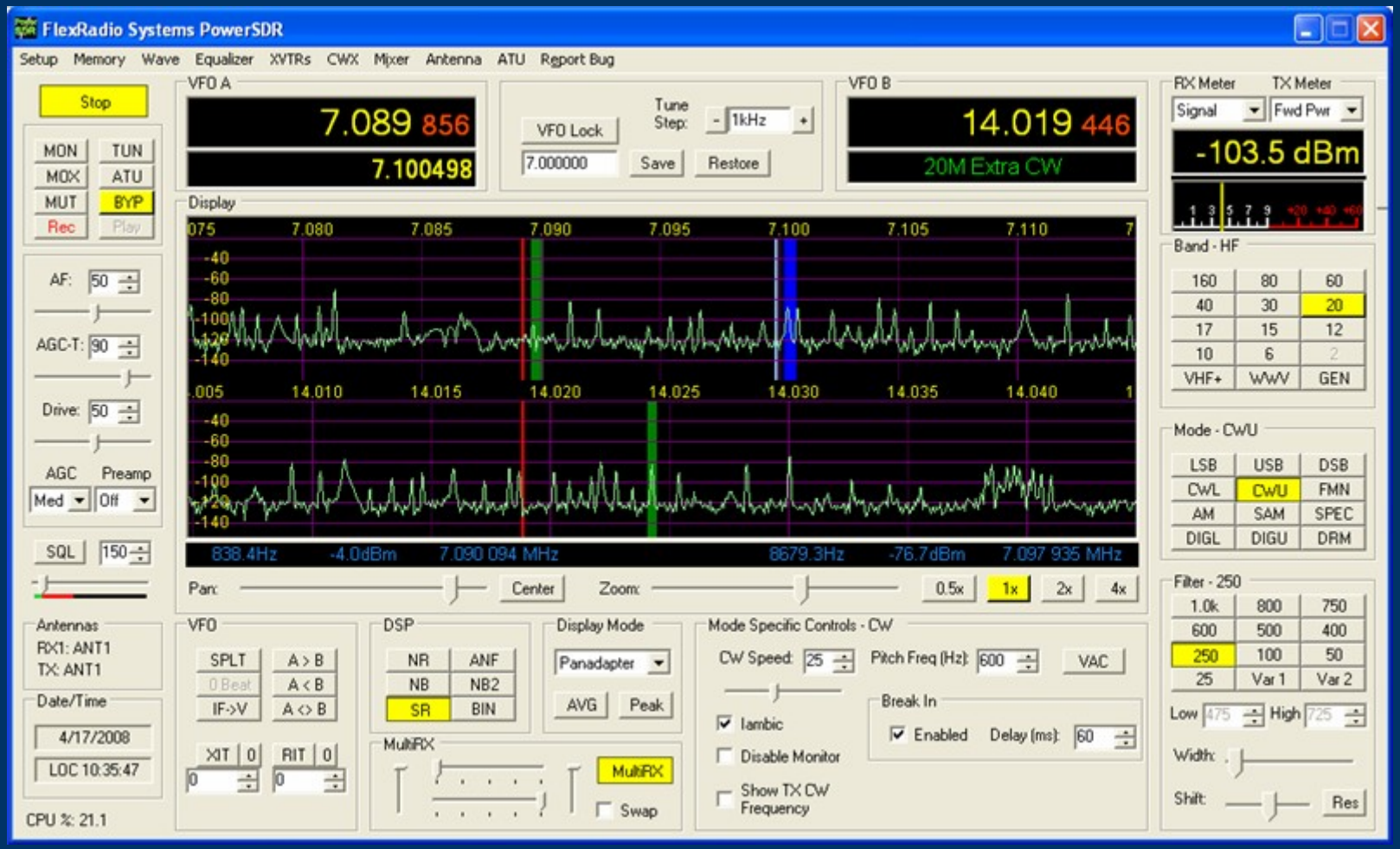
DttSP

Open Source Digital Signal Processing package developed by **Dr.Frank Brickle** and **Dr. Robert McGwier (uses FFTW3).**

Implements the basic modulation, demodulation, signal conditioning, and synchronisation processes required to operate a high performance transceiver using DSP as the detection and synthesis stages.

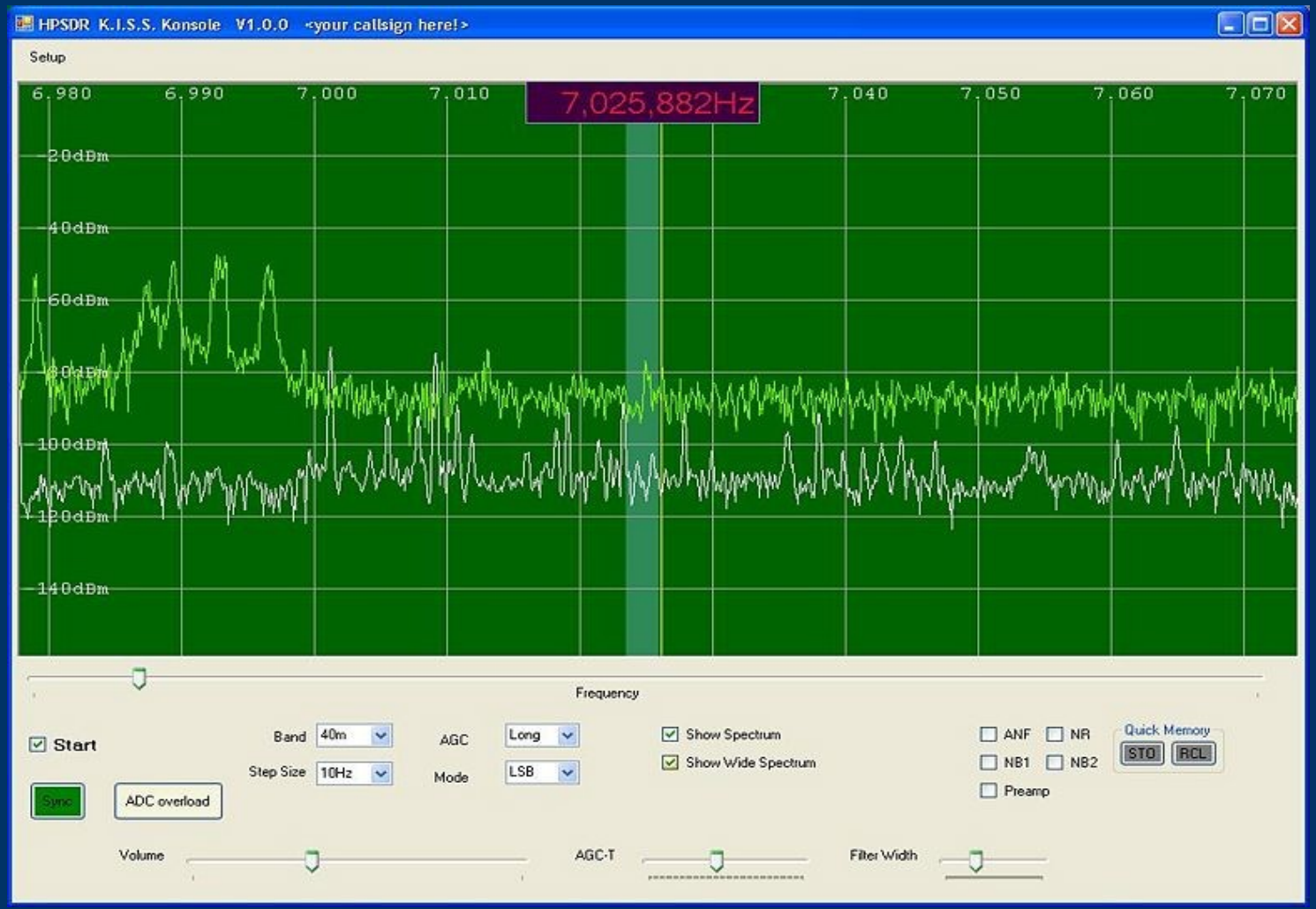
HPSDR

Windows Software – FlexRadio PowerSDR



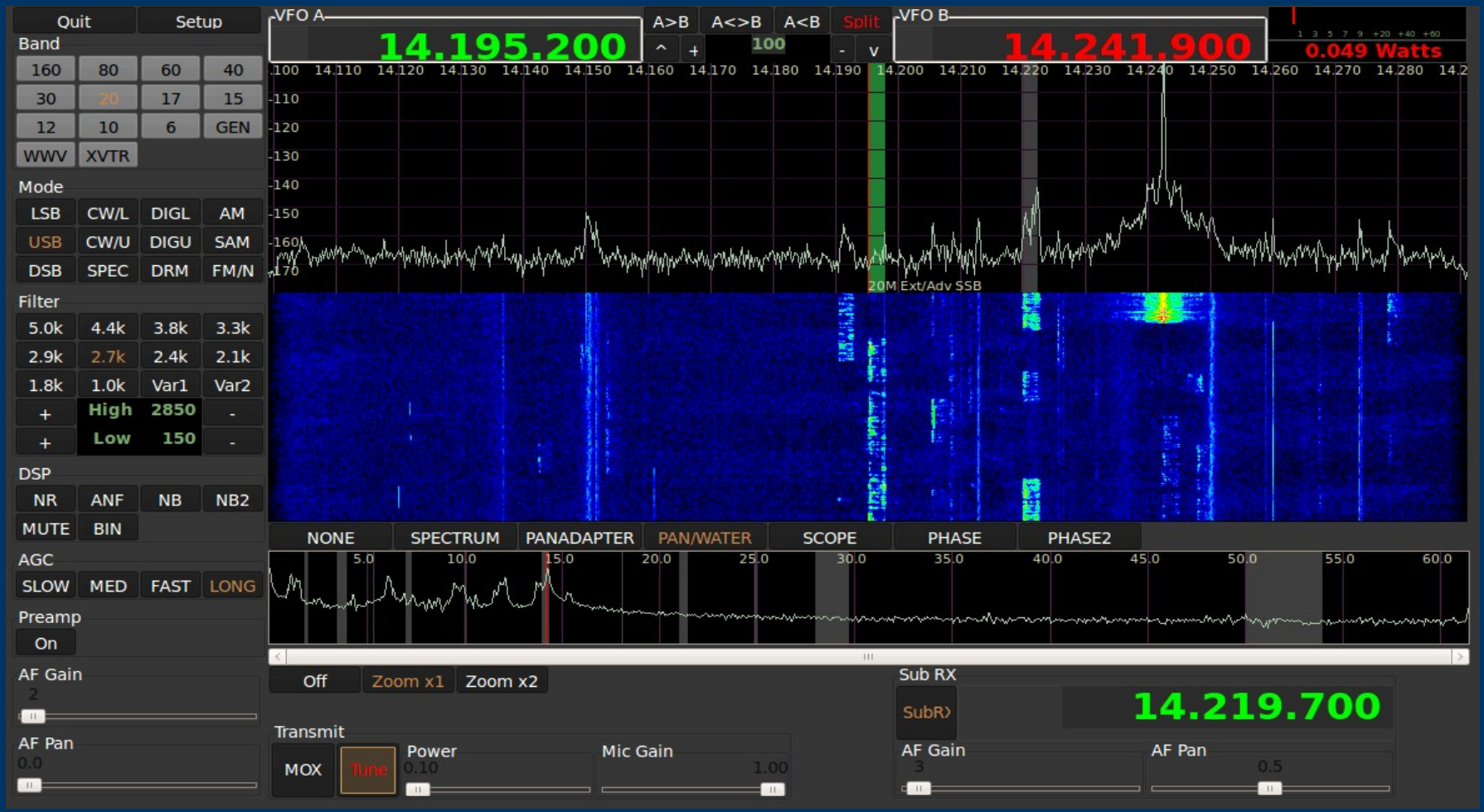
HPSDR

Windows Software – KISS Konsole



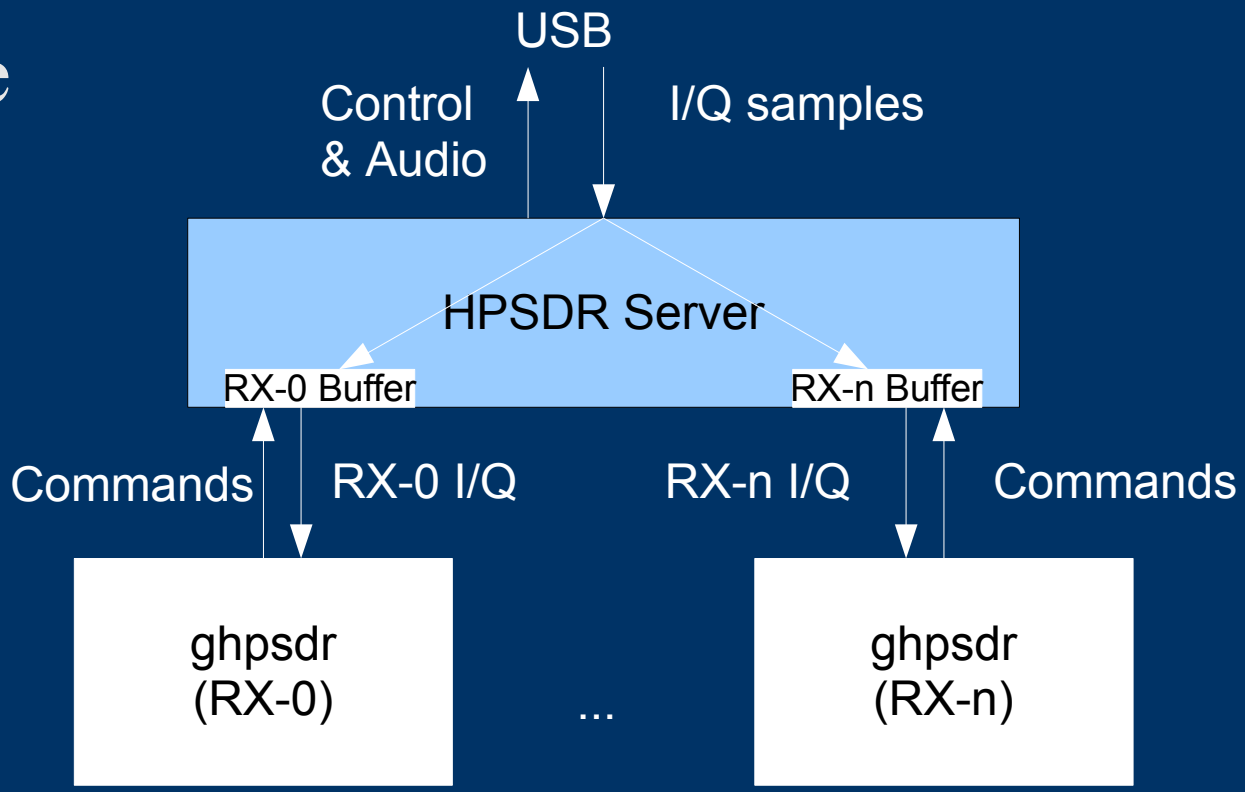
HPSDR

Linux Software - ghpsdr



HPSDR

Software



TCP connection for Commands
I/Q data sent over UDP

USB Interface

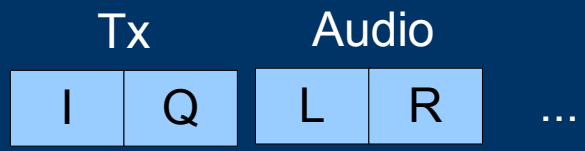
USB Output Data Stream

USB data is transferred in 512 byte packets.
Always at 48K.

8 bytes of header data (rx freq, tx freq, mox, duplex).

Sync	Sync	Sync	C0	C1	C2	C3	C4

63 Transmit I/Q data and Audio samples.



I and Q samples are 16 bits.
Audio samples are 16 bits.

HPSDR

- Software

USB I/Q Data Stream

USB data is transferred in 512 byte packets.

8 bytes of header data.

Sync	Sync	Sync	C0	C1	C2	C3	C4

Receiver I/Q data and Mic sample.

I and Q samples are 24 bits (3 bytes).
Microphone samples are 16 bits (2 bytes).

HPSDR

- Software

Bandwidth Requirements

24 bit I samples
24 bit Q samples

48000 samples per second = 2304000 bits per second
96000 samples per second = 4608000 bits per second
192000 samples per second = 9216000 bits per second

For just 1 receiver!

8 receivers at 192000 = 73728000 bits per second

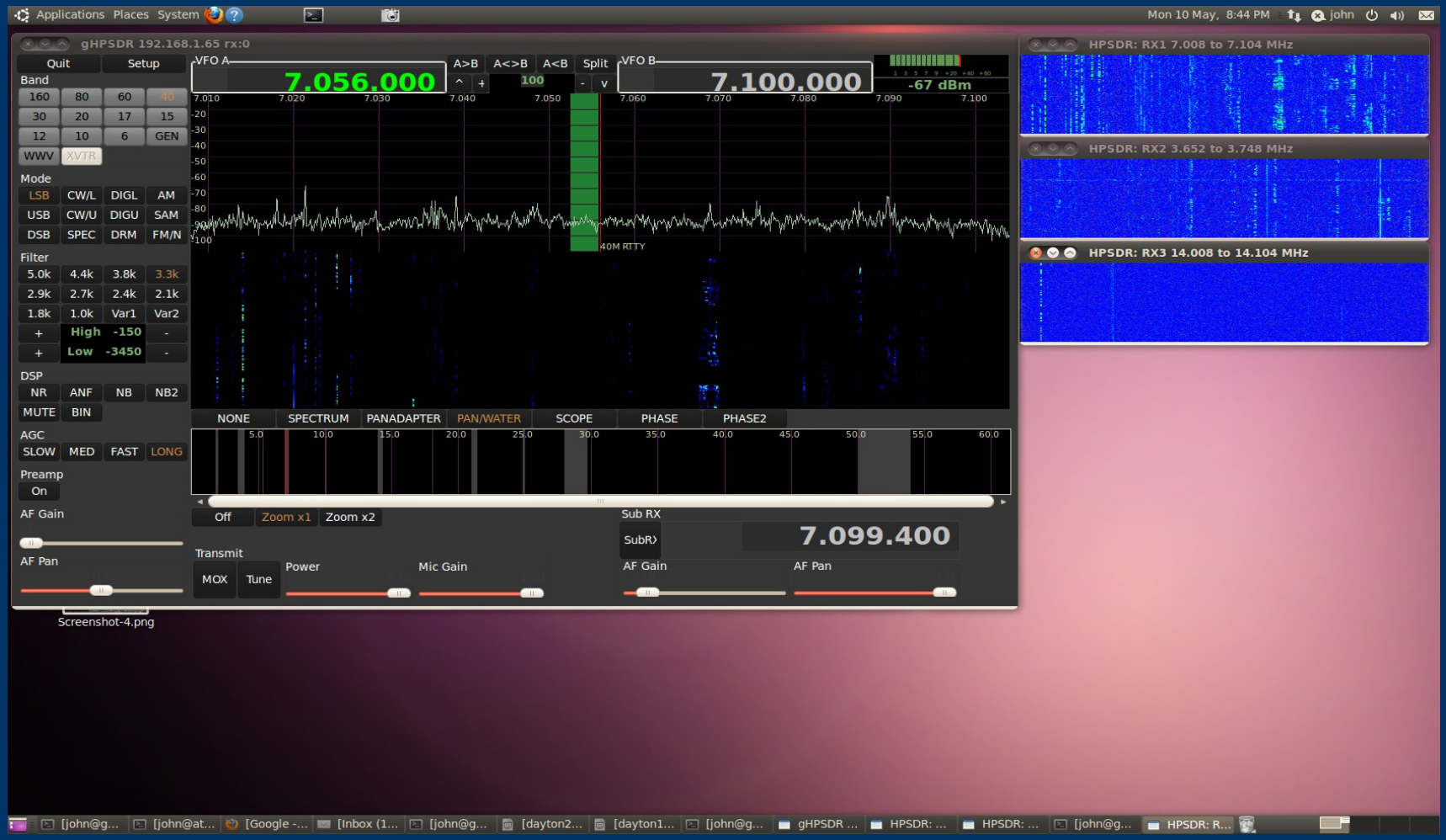
My broadband uplink speed is 360 Kbps

HPSDR

Software

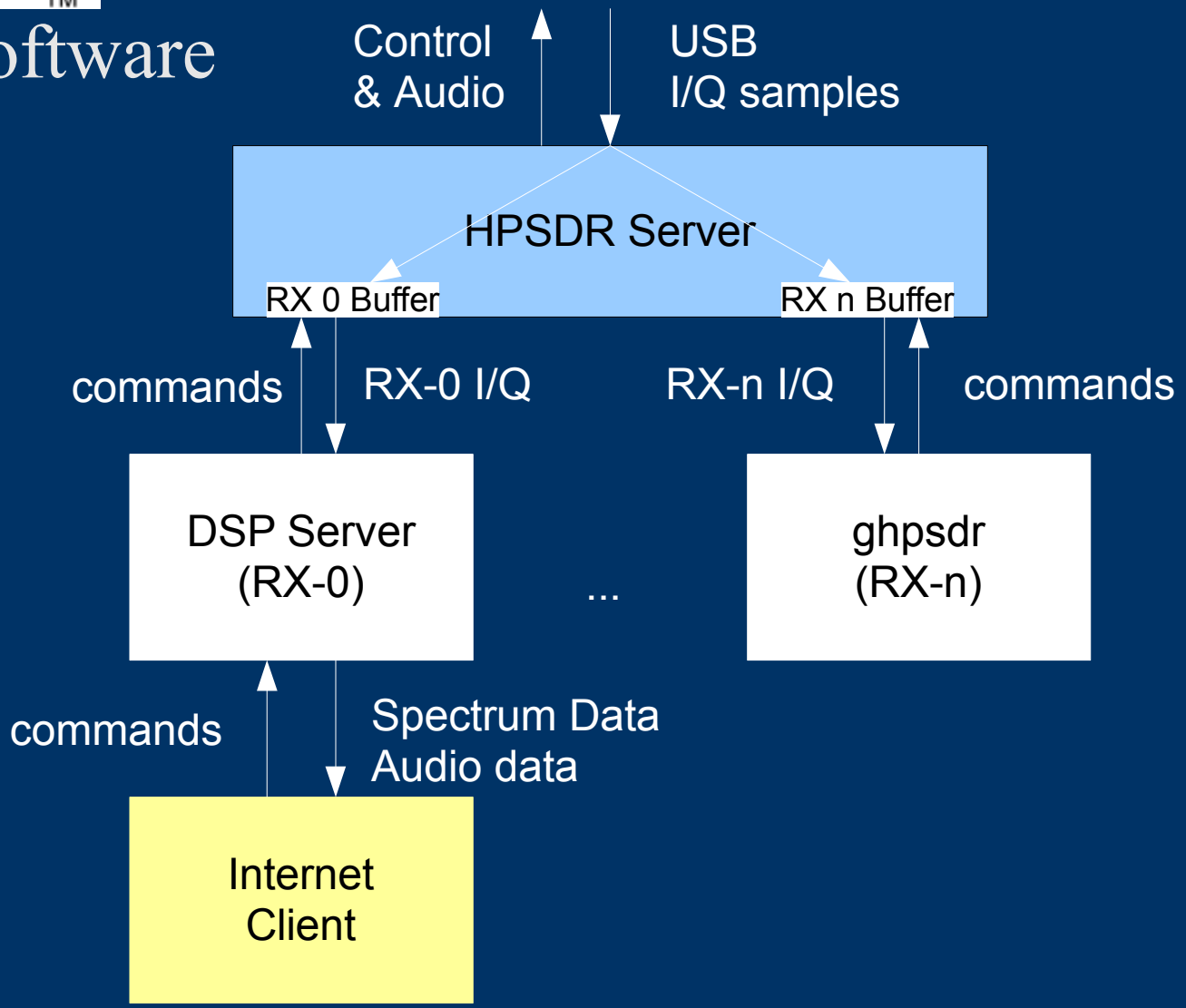
HPSDR

Software



HPSDR

- Software



HPSDR

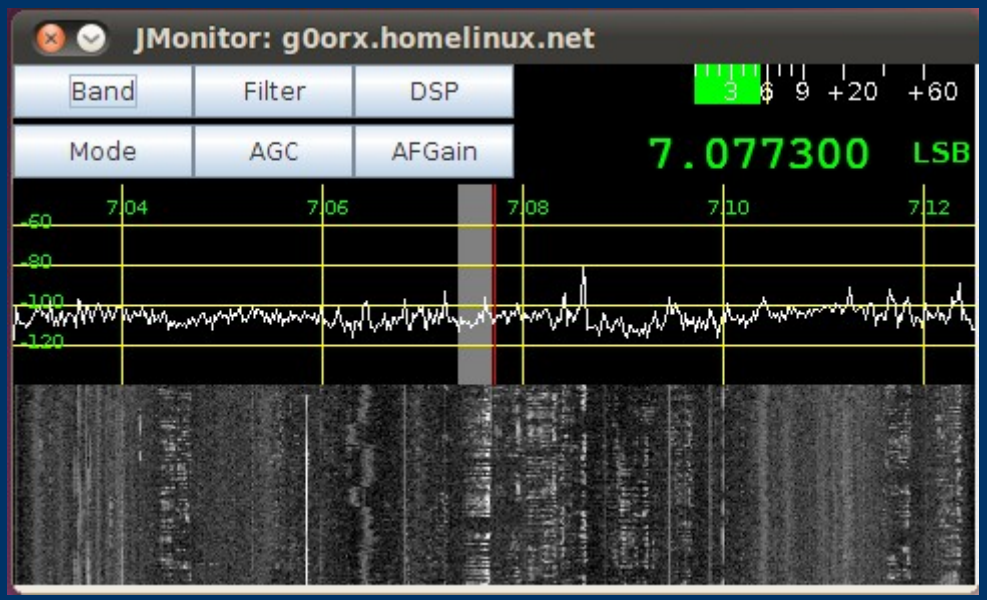
- Software
  - Bandwidth requirements per receiver
    - Spectrum data sent as 8 bit values
    - Client requests number of samples - currently 480 at 10 requests per second.
    - Audio data sent as 8-bit aLaw at 8000 samples per second.
    - Spectrum data = 480*10*8 = 34800 bits per second
    - Audio data =8000*8 = 64000 bits per second
    - 98800 bits per second per receiver

HPSDR

Software

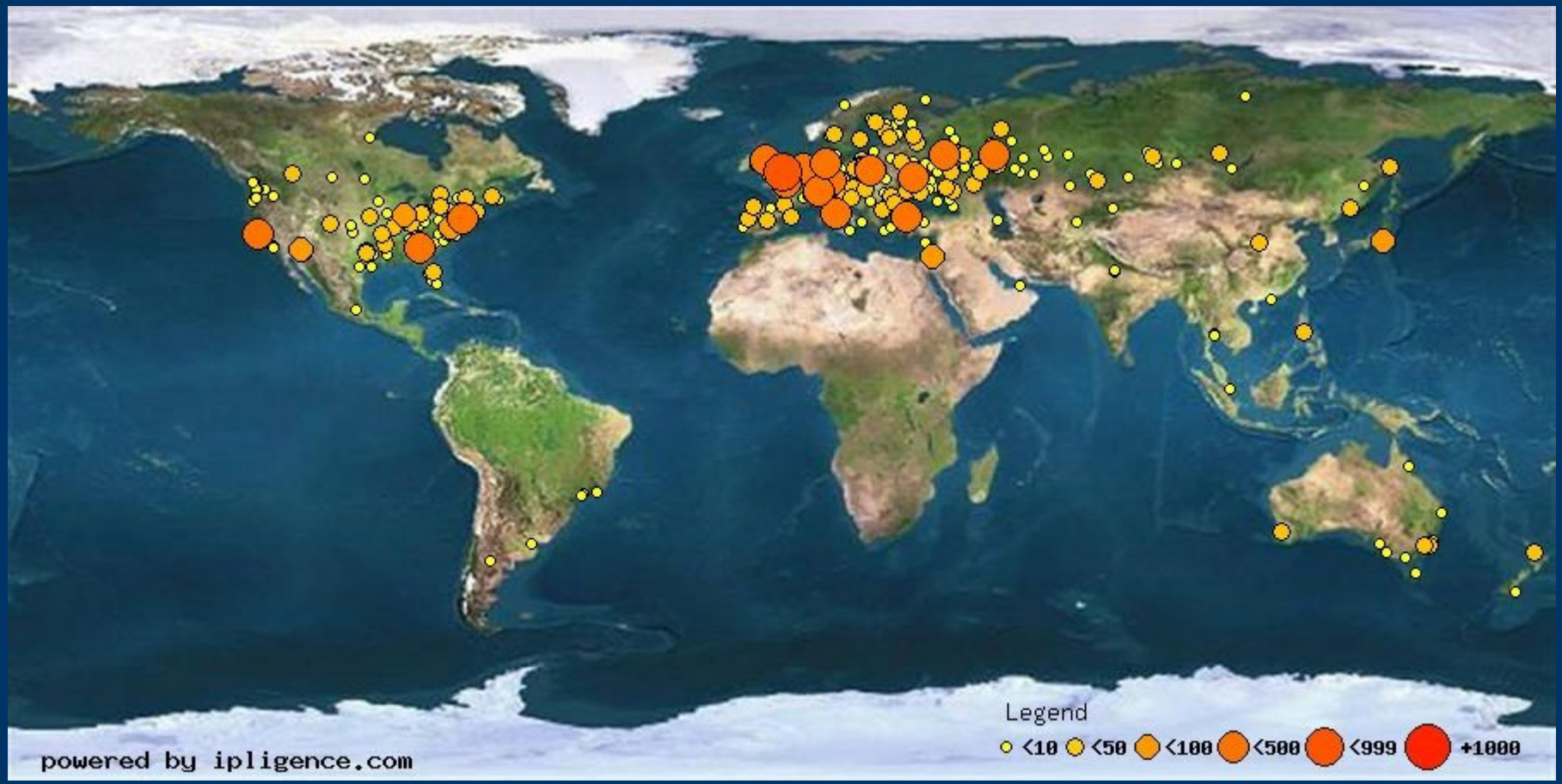
Java Applet Client

- Runs in any browser window.
- Can be run as a standalone application.



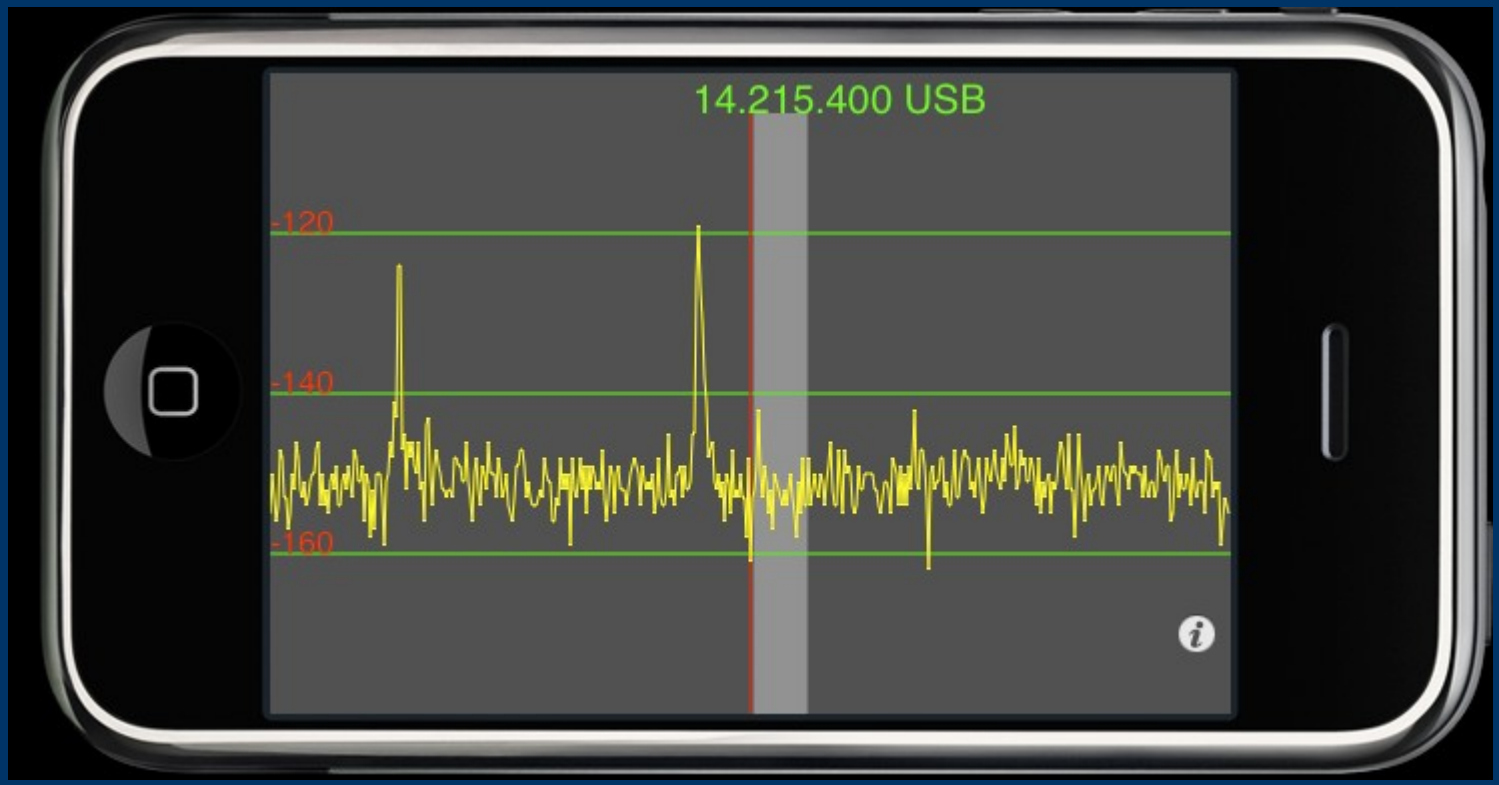
HPSDR

Many accesses from around the world to my web page



HPSDR

Software - iPhone client

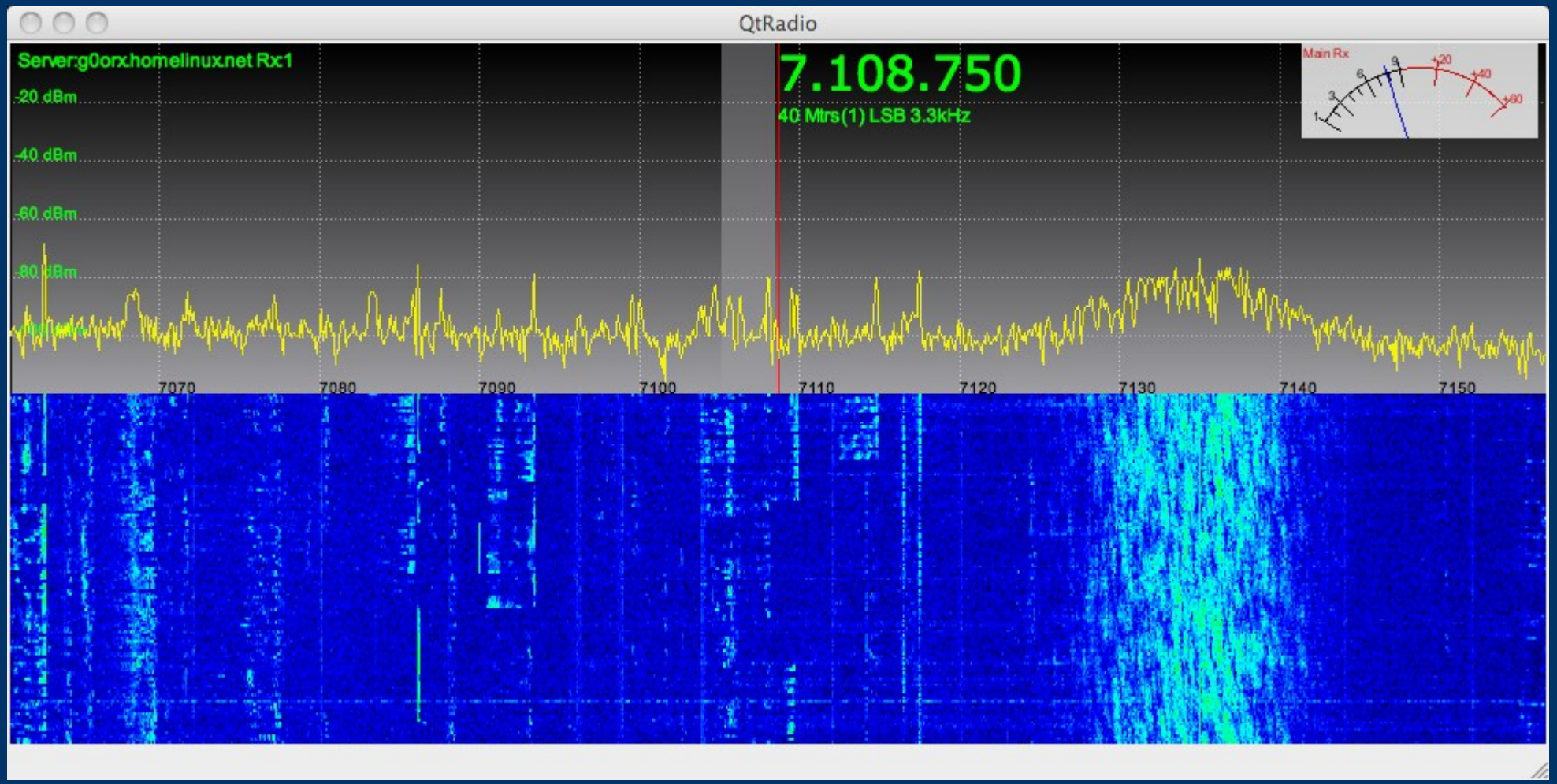


HPSDR

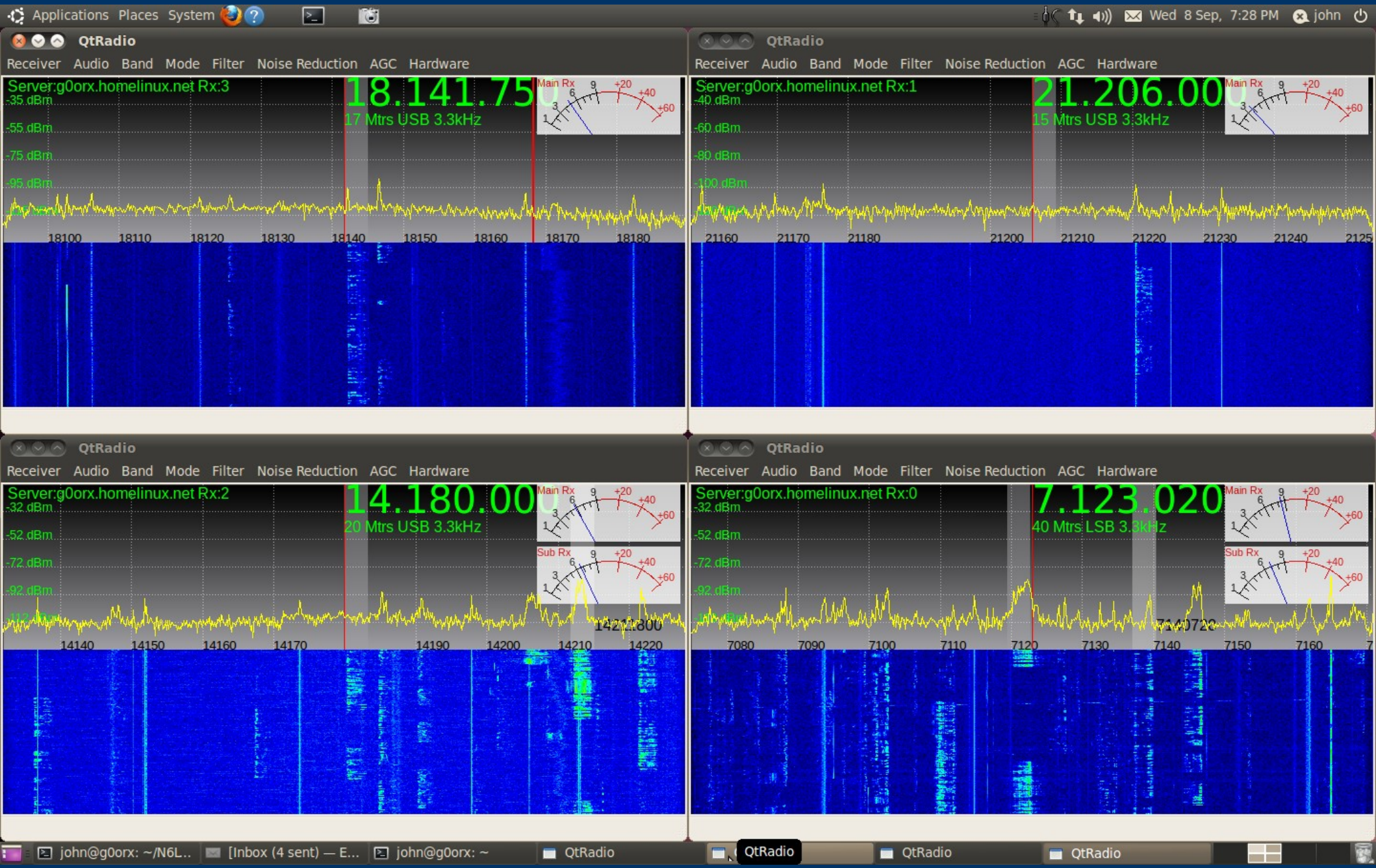
Software - Android Client

HPSDR

Software - QtRadio

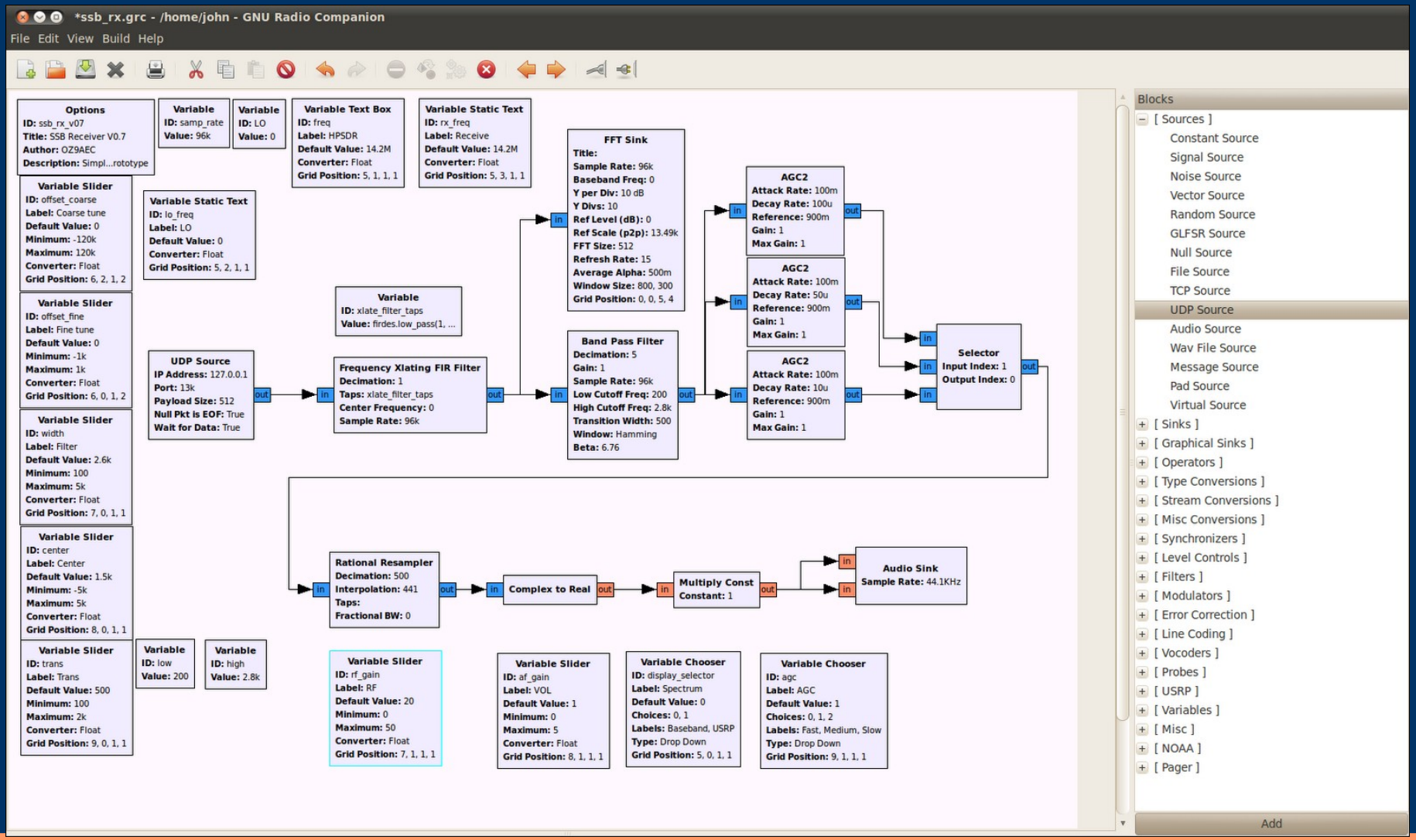


Uses Qt 4 (from Nokia) – multiplatform GUI support


Applications Places System
Wed 8 Sep, 7:28 PM john
QtRadio
Receiver Audio Band Mode Filter Noise Reduction AGC Hardware
Server:g0orx.homelinux.net Rx:3
18.141.750
17 Mtrs USB 3.3kHz
Main Rx
-35 dBm
-55 dBm
-75 dBm
-95 dBm
18100 18110 18120 18130 18140 18150 18160 18170 18180
QtRadio
Receiver Audio Band Mode Filter Noise Reduction AGC Hardware
Server:g0orx.homelinux.net Rx:1
21.206.000
15 Mtrs USB 3.3kHz
Main Rx
-40 dBm
-60 dBm
-80 dBm
-100 dBm
21160 21170 21180 21200 21210 21220 21230 21240 2125
QtRadio
Receiver Audio Band Mode Filter Noise Reduction AGC Hardware
Server:g0orx.homelinux.net Rx:2
14.180.000
20 Mtrs USB 3.3kHz
Main Rx
Sub Rx
-32 dBm
-52 dBm
-72 dBm
-92 dBm
14140 14150 14160 14170 14190 14200 14210 14220
QtRadio
Receiver Audio Band Mode Filter Noise Reduction AGC Hardware
Server:g0orx.homelinux.net Rx:0
7.123.020
40 Mtrs LSB 3.3kHz
Main Rx
Sub Rx
-32 dBm
-52 dBm
-72 dBm
-92 dBm
7080 7090 7100 7110 7120 7130 7140 7150 7160
john@g0orx: ~/N6L..
[Inbox (4 sent) — E...
john@g0orx: ~
QtRadio
QtRadio
QtRadio
QtRadio

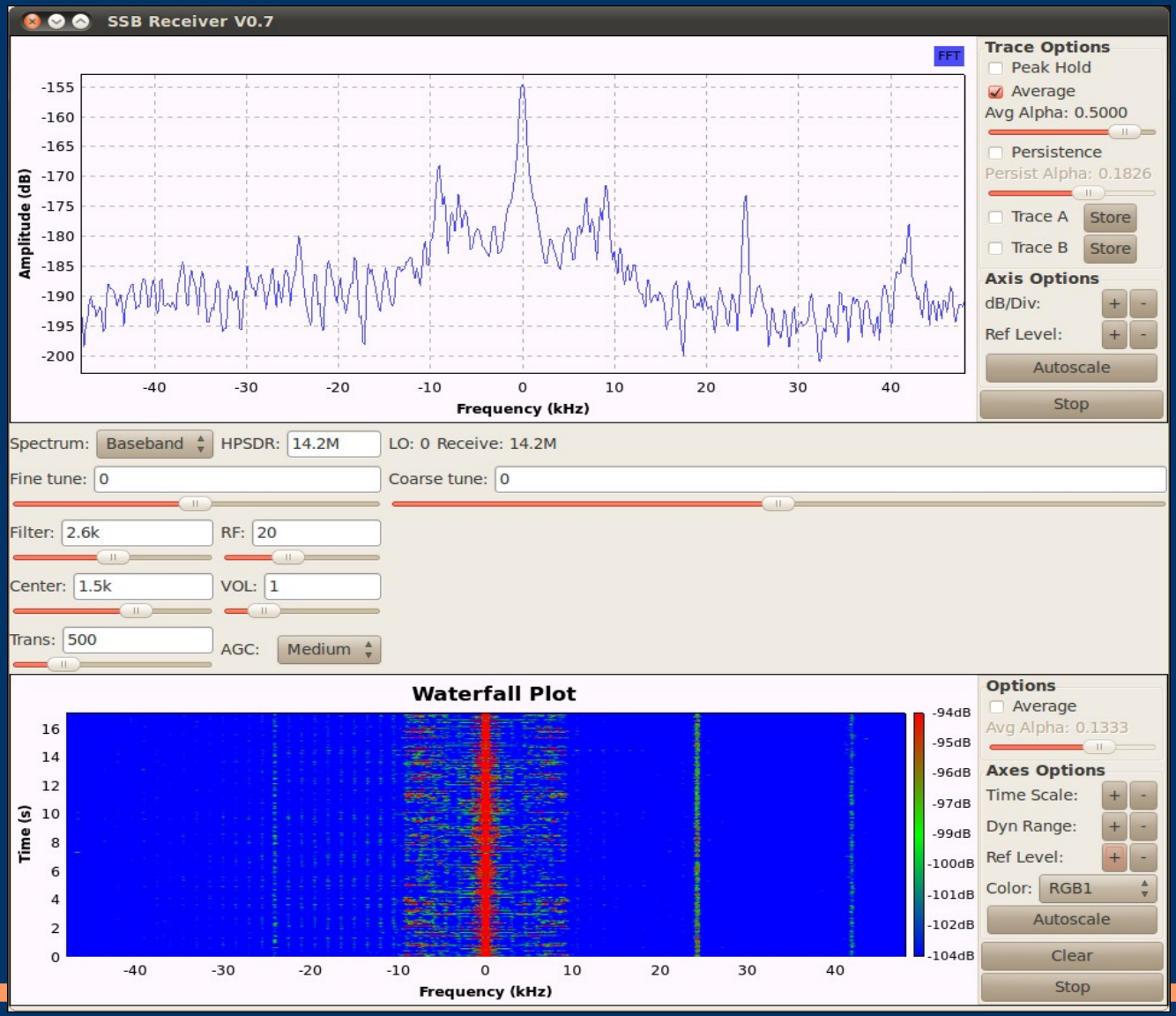
HPSDR

Software – GNU Radio



HPSDR

Software – GNU Radio



HPSDR

Software - QtRadio

Demonstration?

HPSDR References

http://openhpsdr.org
- Wiki
- Teamspeak
- Reflector
- Links

Teamspeak recordings
- http://www.hamsdr.com/dnld.aspx

http://www.tapr.org

svn://64.245.179.219/svn/repos_sdr_hpsdr/trunk/N6LYT

g0orx.blogspot.com

john.d.melton@googlemail.com